R-TYPE(for register)

RISC-V Fields

RISC-V fields are given names to make them easier to discuss:

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

Here is the meaning of each name of the fields in RISC-V instructions:

- ***opcode:*** Basic operation of the instruction, and this abbreviation is its traditional name.
- *rd:* The register destination operand. It gets the result of the operation.
- *funct3:* An additional opcode field.
- *rs1:* The first register source operand.
- *rs2:* The second register source operand.
- *funct7:* An additional opcode field.

opcode The field that denotes the operation and format of an instruction.

I-TYPE(with immediate)

The compromise chosen by the RISC-V designers is to keep all instructions the same length, thereby requiring distinct instruction formats for different kinds of instructions. For example, the format above is called *R-type* (for register). A second type of instruction format is *I-type* and is used by arithmetic operands with one constant operand, including `addi`, and by load instructions. The fields of the I-type format are

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

The 12-bit immediate is interpreted as a two's complement value, so it can represent integers from -2^{11} to $2^{11}-1$. When the I-type format is used for load instructions, the immediate represents a byte offset, so the load word instruction can refer to any word within a region of $\pm 2^{11}$ or 2048 bytes ($\pm 2^{8}$ or 512 words) of the base address in the base register rd. We see that more than 32 registers would be difficult in this format, as the rd and rs1 fields would each need another bit, making it harder to fit everything in one word.

S-TYPE(Store)

and 9 (for x9) is placed in the rd field. We also need a format for the store word instruction, sw, which needs two source registers (for the base address and the store data) and an immediate for the address offset. The fields of the S-type format are

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

The 12-bit immediate in the S-type format is split into two fields, which supply the lower 5 bits and upper 7 bits. The RISC-V architects chose this design because it keeps the rs1 and rs2 fields in the same place in all instruction formats.

Branches

Format: {comparison} {reg1} {reg2} {label}

If we don't branch

- PC = PC + 4

If we do branch

- PC = PC + immediate

What range of instructions can we branch to?

- 2's complement range: $[-2^{n-1}, 2^{n-1}-1]$
- With 12 bits: $\pm 2^{11}$ bytes away from the PC
- Instructions are 4-bytes, so we can jump $\pm 2^{9}$ instructions away from the current instruction

0x10000000	Instruction 0
0x10000004	Instruction 1
0x10000008	Instruction 2
0x1000000C	Instruction 3
0x10000010	Instruction 4
0x10000014	Instruction 5
0x10000018	Instruction 6
0x1000001C	Instruction 7

The last nibble is always `0x(0, 4, 8, or C)`

```
0x0 = 0b0000
0x4 = 0b0100
0x8 = 0b1000
0xC = 0b1100
```

The last two bits are always `0b00`

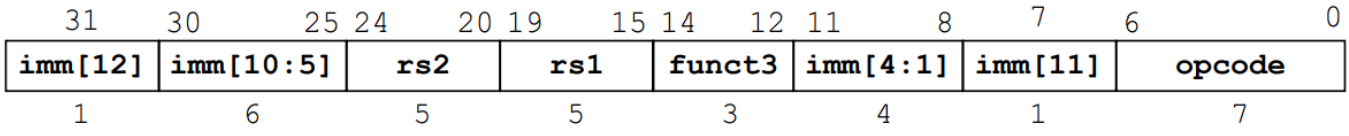
- The last two bits are always 0, so we can increase our range by not storing those bits

encoded immediate = 0b0000 0000 0011

actual immediate offset = 0b00 0000 0000 1100

- (PC + immediate) will go 3 instructions (or 12 bytes) away
- Now, we can jump $\pm 2^{11}$ instructions (or $\pm 2^{13}$ bytes) away from the current PC

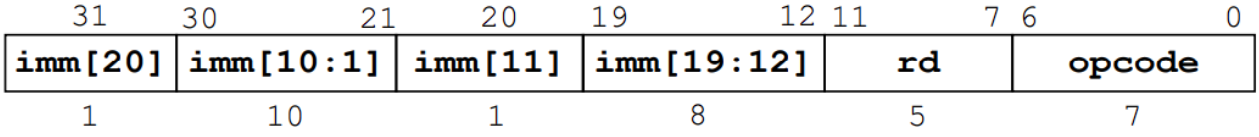
Branch Format



J-Format

`jal rd, Label` ← The label that we want to jump to

rd = register where the return address will be stored



U-Format

LUI to create long immediates

- LUI writes the upper 20 bits of the destination with the immediate value, and clears the lower 12 bits.
- Together with an ADDI to set low 12 bits, can create any 32-bit value in a register using two instructions (LUI/ADDI).

```
LUI x10, 0x87654        # x10 = 0x87654000
ADDI x10, x10, 0x321    # x10 = 0x87654321
```


Pseudo-instruction that performs lui and addi

```
LUI x10, 0x87654        # x10 = 0x87654000
ADDI x10, x10, 0x321    # x10 = 0x87654321
```

```
LI x10, 0x87654321      # x10 = 0x87654321
```

- If the value being added is negative, add 1 to the upper 20 bits before adding the 12-bit value
- How to set 0xABCDEEEE?

```
lui x10, ABCDF   # x10 = 0xABCDF000
addi x10, 0xEEE  # x10 = 0xABCDEEEE
```

- `li` instruction will automatically handle this corner case

Add Upper Immediate PC (AUIPC)

- rd = PC + (immediate << 12)

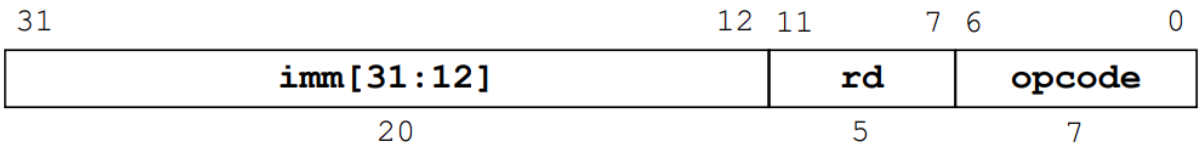
```
auipc x5, 0xABCDE
```

Destination register

Immediate value

```
x5 = PC + 0xABCDE000
```

U-Format for Upper Immediate Instructions

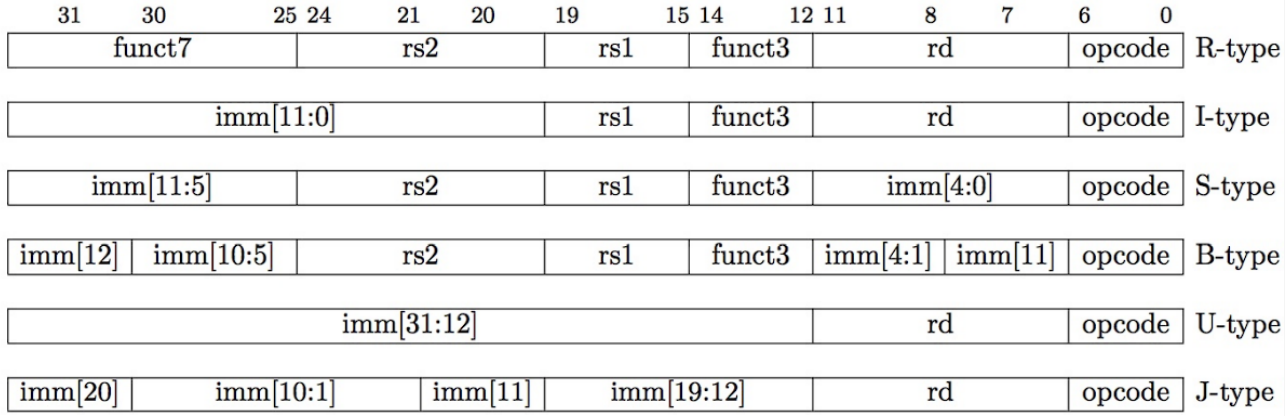


Summary

31 25	24 20	19 15	14 12	11 7	6 0
R	funct7	rs2	rs1	funct3	rd	opcode
I	imm[11:0]		rs1	funct3	rd	opcode
I*	funct7	imm[4:0]	rs1	funct3	rd	opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
B	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]				rd	opcode
J	imm[20|10:1|11|19:12]				rd	opcode

Immediates are sign-extended to 32 bits, except in I* type instructions and **`sltiu`**.

Summary of RISC-V Instruction Formats



Type	ImmSel (default)	Bits 31-20	Bits 19-12	Bit 11	Bits 10-5	Bits 4-1	Bit 0
I	`0b000`	`inst[31]`			`inst[30:20]`		
S	`0b001`	`inst[31]`			`inst[30:25]`	`inst[11:7]`	
B	`0b010`	`inst[31]`		`inst[7]`	`inst[30:25]`	`inst[11:8]`	`0`
U	`0b011`	`inst[31:12]`		`0`			
J	`0b100`	`inst[31]`	`inst[19:12]`	`inst[20]`	`inst[30:21]`		`0`