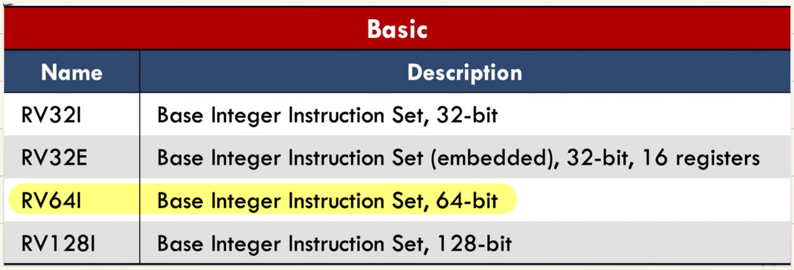
RISC-V ISA 分為：



Basic	
Name	**Description**
RV32I	Base Integer Instruction Set, 32-bit
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers
RV64I	Base Integer Instruction Set, 64-bit
RV128I	Base Integer Instruction Set, 128-bit

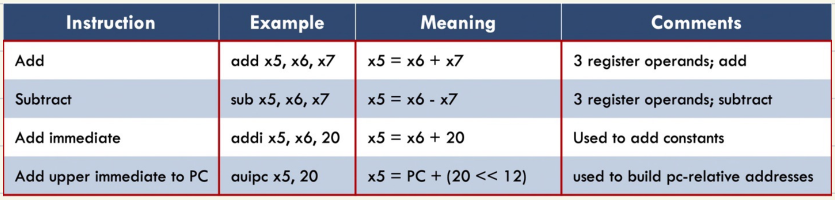
ISA 需包含：

①. Instruction set

②. Hardware information：
- I. Register
- II. Memory
- III. addressing mode
- IV. instruction format

Instruction Set：指令分為：
①. Arithmetic
②. Data transfer
③. Logical
④. Shift
⑤. Conditional Branch
⑥. Unconditional Branch

①. Arithmetic：



Instruction	Example	Meaning	Comments
Add	add x5, x6, x7	x5 = x6 + x7	3 register operands; add
Subtract	sub x5, x6, x7	x5 = x6 - x7	3 register operands; subtract
Add immediate	addi x5, x6, 20	x5 = x6 + 20	Used to add constants
Add upper immediate to PC	auipc x5, 20	x5 = PC + (20 << 12)	used to build pc-relative addresses

@. Data transfer :

Instruction	Example	Meaning	Comments
Load doubleword	ld x5, 40(x6)	x5 = Mem [x6 + 40]	Doubleword from Mem. to Reg.
Store doubleword	sd x5, 40(x6)	Mem [x6 + 40] = x5	Doubleword from Reg. to Mem.
Load word	lw x5, 40(x6)	x5 = Mem [x6 + 40]	Word from Mem. to Reg.
Load word unsigned	lwu x5, 40(x6)	x5 = Mem [x6 + 40]	Unsigned word from Mem. to Reg.
Store word	sw x5, 40(x6)	Mem [x6 + 40] = x5	Word from Reg. to Mem.
Load halfword	lh x5, 40(x6)	x5 = Mem [x6 + 40]	Halfword from Mem. to Reg.
Load halfword unsigned	lhu x5, 40(x6)	x5 = Mem [x6 + 40]	Unsigned halfword from Mem. to Reg.
Store halfword	sh x5, 40(x6)	Mem [x6 + 40] = x5	Halfword from Reg. to Mem.
Load byte	lb x5, 40(x6)	x5 = Mem [x6 + 40]	Byte from Mem. to Reg.
Load byte unsigned	lbu x5, 40(x6)	x5 = Mem [x6 + 40]	Unsigned byte from Mem. to Reg.
Store byte	sb x5, 40(x6)	Mem [x6 + 40] = x5	Byte from Reg. to Mem.
Load upper immediate	lui x5, 0x12345	x5 = 0x12345000	Load 20 bit constant shifted left 12 bits

@. logical :

Instruction	Example	Meaning	Comments
And	and x5, x6, x7	x5 = x6 & x7	3 reg. operands; bit-by-bit AND
Inclusive or	or x5, x6, x7	x5 = x6 | x7	3 reg. operands; bit-by-bit OR
Exclusive or	xor x5, x6, x7	x5 = x6 ^ x7	3 reg. operands; bit-by-bit XOR
And immediate	andi x5, x6, 20	x5 = x6 & 20	Bit-by-bit AND reg. with constant
Inclusive or immediate	ori x5, x6, 20	x5 = x6 | 20	Bit-by-bit OR reg. with constant
Exclusive or immediate	xori x5, x6, 20	x5 = x6 ^ 20	Bit-by-bit XOR reg. with constant

@ shift :

Instruction	Example	Meaning	Comments
Shift left logic	sll x5, x6, x7	x5 = x6 << x7	Shift left by register
Shift right logic	srl x5, x6, x7	x5 = x6 >> x7	Shift right by register
Shift right arithmetic	sra x5, x6, x7	x5 = x6 >> x7	Arithmetic shift right by register
Shift left logic immediate	sll x5, x6, 3	x5 = x6 << 3	Shift left by immediate
Shift right logic immediate	srl x5, x6, 3	x5 = x6 >> 3	Shift right by immediate
Shift right arithmetic immediate	sra x5, x6, 3	x5 = x6 >> 3	Arithmetic shift right by immediate

⑤. Conditional Branch:

Instruction	Example	Meaning	Comments
Branch if equal	beq x5, x6, 100	If (x5 = x6) go to PC + 100	PC-relative branch if equal
Branch if not equal	bne x5, x6, 100	If (x5 != x6) go to PC + 100	PC-relative branch if not equal
Branch if less than	blt x5, x6, 100	If (x5 < x6) go to PC + 100	PC-relative branch if less
Branch if greater & eq.	bge x5, x6, 100	If (x5 ≥ x6) go to PC + 100	PC-relative branch if greater or equal
Branch if less, unsigned	bltu x5, x6, 100	If (x5 < x6) go to PC + 100	PC-relative branch if less, unsigned
Branch if ≥, unsigned	bgeu x5, x6, 100	If (x5 ≥ x6) go to PC + 100	PC-relative branch if ≥, unsigned

⑥. Unconditional Branch:

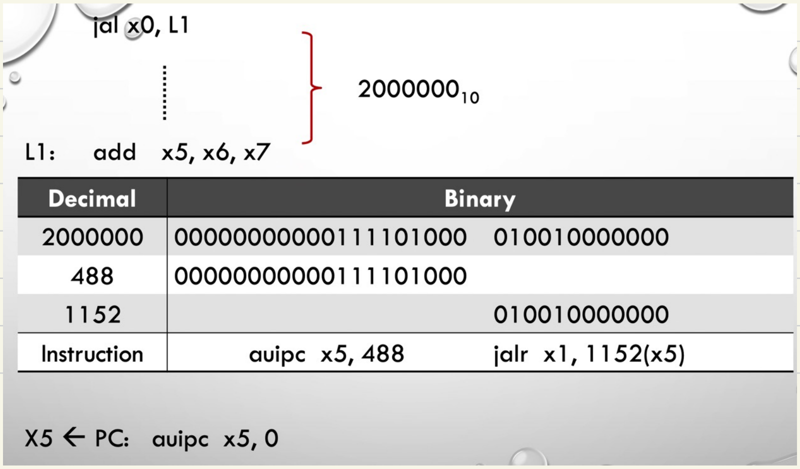
Instruction	Example	Meaning	Comments
Jump & link	jal x1, 100	x1 = PC+4; go to PC + 100	PC-relative procedure call
Jump & link reg.	jalr x1, 100(x5)	x1 = PC+4; go to x5 + 100	Procedure return; indirect call

Instruction Set 相關:

1. Nop 指令為 addi x0, x0, 0 (MIPS 為 sll $s0, $s0, 0)

2. The combination of an AUIPC and the 12-bit immediate in a JALR can transfer control to any 32-bit PC-relative address, while an AUIPC plus the 12-bit immediate offset in regular load or store instructions can access any 32-bit PC-relative data address.

3. The current PC can be obtained by setting the U-immediate to 0



```
      jal x0, L1
                      }  2000000₁₀
L1:   add  x5, x6, x7
```

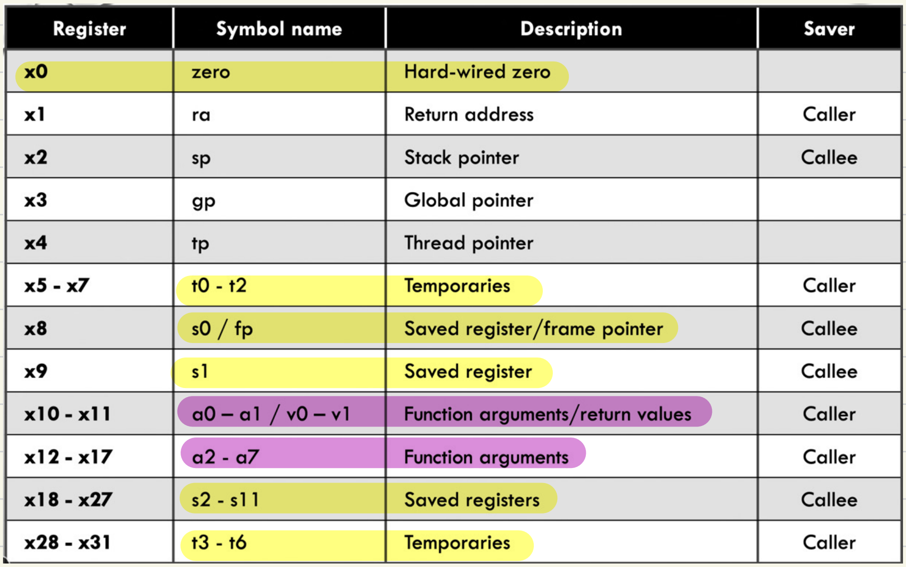
Decimal	Binary	
2000000	00000000000111101000	010010000000
488	00000000000111101000	
1152		010010000000
Instruction	auipc x5, 488	jalr x1, 1152(x5)

X5 ← PC: auipc x5, 0

I. Register: RISC-V 共有 32个 64 bit 的 register file (x0 ~ x31)

32-bit data 為 word

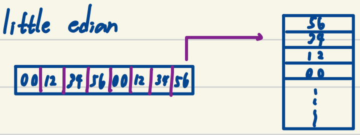
64-bit data 為 double word

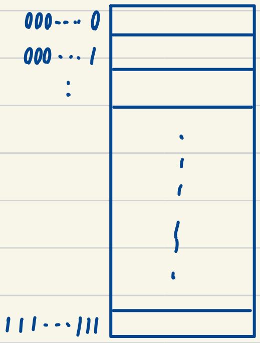


Register	Symbol name	Description	Saver
x0	zero	Hard-wired zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5 - x7	t0 - t2	Temporaries	Caller
x8	s0 / fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10 - x11	a0 – a1 / v0 – v1	Function arguments/return values	Caller
x12 - x17	a2 - a7	Function arguments	Caller
x18 - x27	s2 - s11	Saved registers	Callee
x28 - x31	t3 - t6	Temporaries	Caller

II. Memory:

1. data structure 無法 fit in register ⇒ 必須透過 main memory 去 access
   arrays, structures, dynamic data
2. byte-addrecing ⇒ 每个 memory address 為 64-bit
3. mem physical space size: 2^{64} bytes
4. 一个 doubleword access 需 access 八个 address
5. little edian





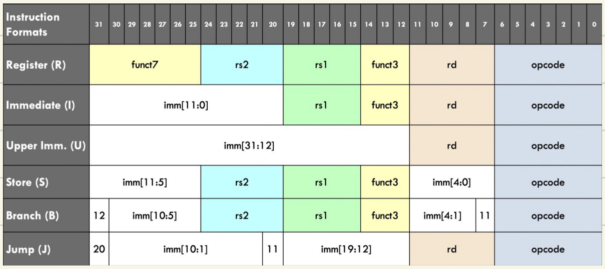
6. 不用 word alignment

Ⅲ. Instruction Format:

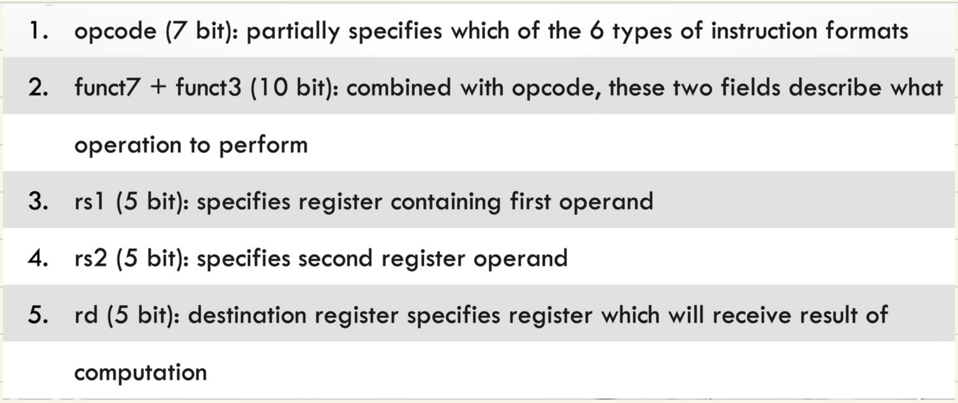
RISC-V instruction 為 32 个 bits 長

∵ 為 little endian $\longleftarrow$

共有:
1. R-type format
2. I-type format
3. U-type format
4. S-type format
5. B-type format
6. J-type format



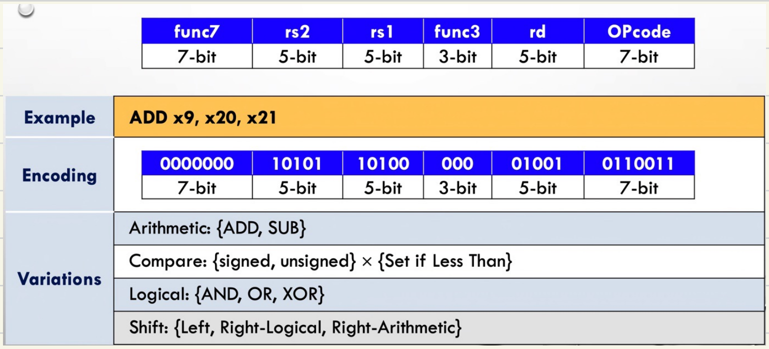
Instruction Formats	31	30 29 28 27 26 25	24 23 22 21	20	19 18 17 16 15	14 13 12	11 10 9 8	7	6 5 4 3 2 1 0
Register (R)	funct7		rs2		rs1	funct3	rd		opcode
Immediate (I)	imm[11:0]				rs1	funct3	rd		opcode
Upper Imm. (U)	imm[31:12]						rd		opcode
Store (S)	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
Branch (B)	12	imm[10:5]	rs2		rs1	funct3	imm[4:1]	11	opcode
Jump (J)	20	imm[10:1]		11	imm[19:12]		rd		opcode



1. opcode (7 bit): partially specifies which of the 6 types of instruction formats
2. funct7 + funct3 (10 bit): combined with opcode, these two fields describe what operation to perform
3. rs1 (5 bit): specifies register containing first operand
4. rs2 (5 bit): specifies second register operand
5. rd (5 bit): destination register specifies register which will receive result of computation

∵ register 編号為 32 bits ⇒ Instruction format 中 register encoding 為 5 bit

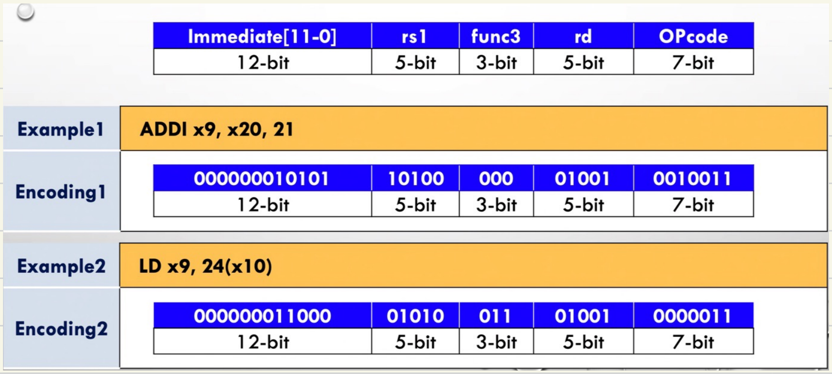
1. R-type format



func7	rs2	rs1	func3	rd	OPcode
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

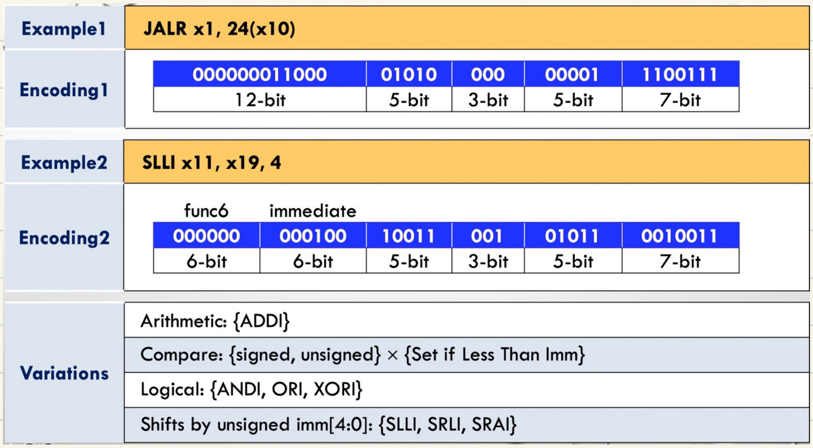
Example	ADD x9, x20, x21
Encoding	0000000 (7-bit) | 10101 (5-bit) | 10100 (5-bit) | 000 (3-bit) | 01001 (5-bit) | 0110011 (7-bit)
Variations	Arithmetic: {ADD, SUB}
	Compare: {signed, unsigned} × {Set if Less Than}
	Logical: {AND, OR, XOR}
	Shift: {Left, Right-Logical, Right-Arithmetic}

2. I-type format



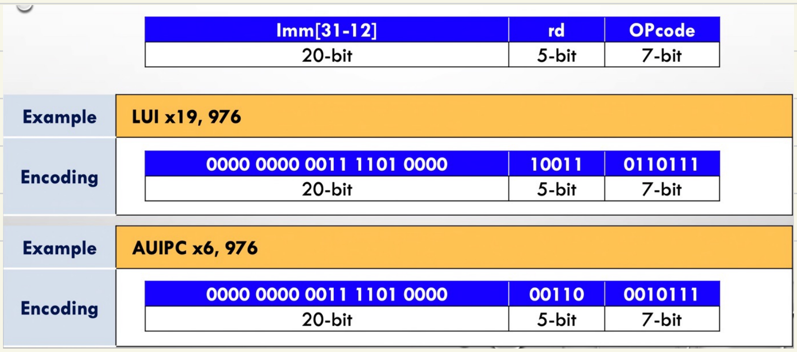
Immediate[11-0]	rs1	func3	rd	OPcode
12-bit	5-bit	3-bit	5-bit	7-bit

Example1	**ADDI x9, x20, 21**
Encoding1	000000010101 (12-bit) | 10100 (5-bit) | 000 (3-bit) | 01001 (5-bit) | 0010011 (7-bit)
Example2	**LD x9, 24(x10)**
Encoding2	000000011000 (12-bit) | 01010 (5-bit) | 011 (3-bit) | 01001 (5-bit) | 0000011 (7-bit)



Example1	**JALR x1, 24(x10)**
Encoding1	000000011000 (12-bit) | 01010 (5-bit) | 000 (3-bit) | 00001 (5-bit) | 1100111 (7-bit)
Example2	**SLLI x11, x19, 4**
Encoding2	func6 000000 (6-bit) | immediate 000100 (6-bit) | 10011 (5-bit) | 001 (3-bit) | 01011 (5-bit) | 0010011 (7-bit)
Variations	Arithmetic: {ADDI}
	Compare: {signed, unsigned} × {Set if Less Than Imm}
	Logical: {ANDI, ORI, XORI}
	Shifts by unsigned imm[4:0]: {SLLI, SRLI, SRAI}

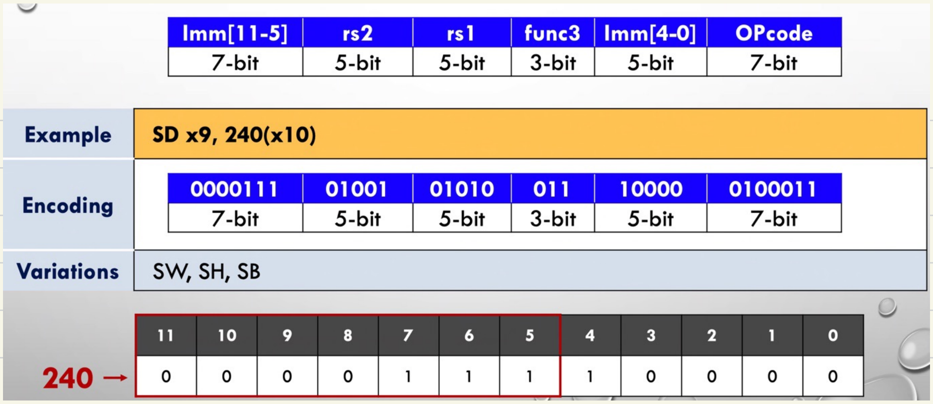
3. U-type format



Imm[31-12]	rd	OPcode
20-bit	5-bit	7-bit

Example	**LUI x19, 976**
Encoding	0000 0000 0011 1101 0000 (20-bit) | 10011 (5-bit) | 0110111 (7-bit)
Example	**AUIPC x6, 976**
Encoding	0000 0000 0011 1101 0000 (20-bit) | 00110 (5-bit) | 0010111 (7-bit)

4. S-type format

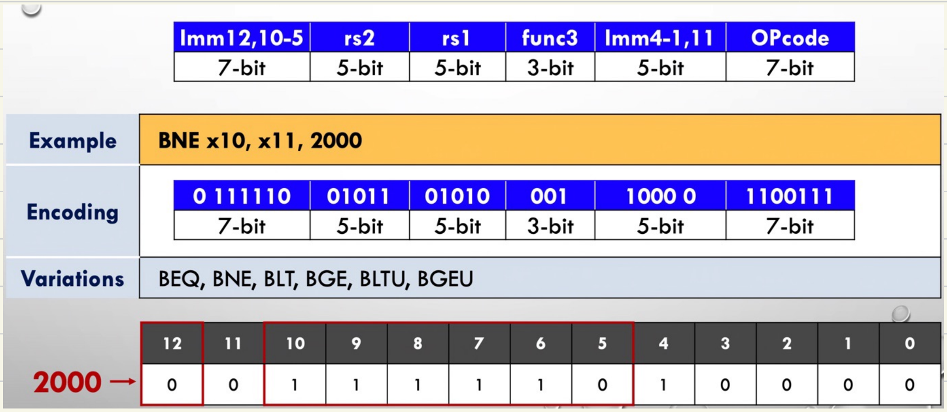


Imm[11-5]	rs2	rs1	func3	Imm[4-0]	OPcode
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

Example	SD x9, 240(x10)
Encoding	0000111 (7-bit), 01001 (5-bit), 01010 (5-bit), 011 (3-bit), 10000 (5-bit), 0100011 (7-bit)
Variations	SW, SH, SB

	11	10	9	8	7	6	5	4	3	2	1	0
240 →	0	0	0	0	1	1	1	1	0	0	0	0

5. B-type format

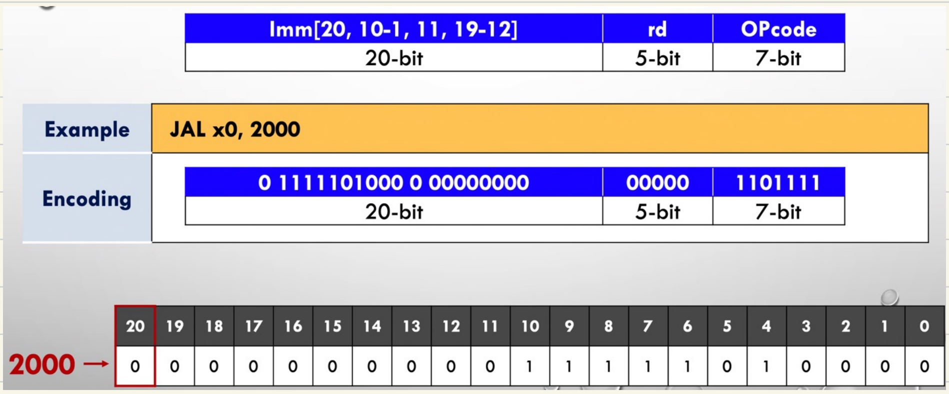


Imm12,10-5	rs2	rs1	func3	Imm4-1,11	OPcode
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

Example	BNE x10, x11, 2000
Encoding	0 111110 (7-bit), 01011 (5-bit), 01010 (5-bit), 001 (3-bit), 1000 0 (5-bit), 1100111 (7-bit)
Variations	BEQ, BNE, BLT, BGE, BLTU, BGEU

	12	11	10	9	8	7	6	5	4	3	2	1	0
2000 →	0	0	1	1	1	1	1	0	1	0	0	0	0

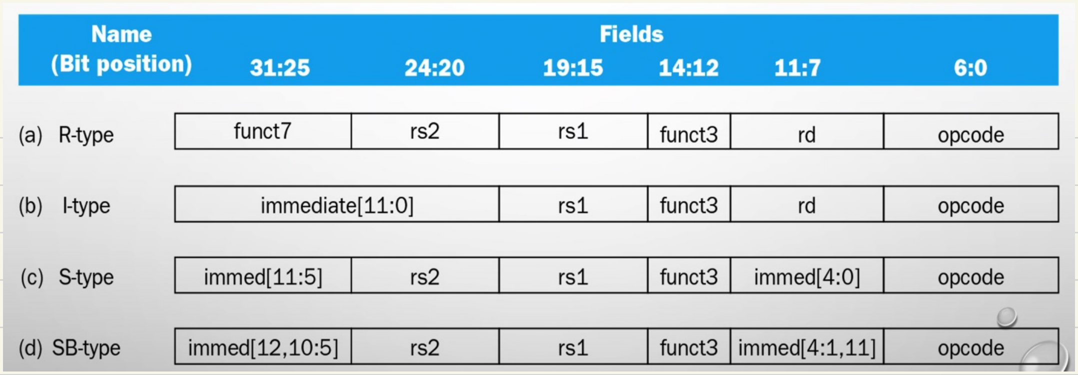
6. J-type format



Imm[20, 10-1, 11, 19-12]	rd	OPcode
20-bit	5-bit	7-bit

Example	JAL x0, 2000
Encoding	0 1111101000 0 00000000 (20-bit), 00000 (5-bit), 1101111 (7-bit)

	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2000 →	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0



Name	Fields					
(Bit position)	31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode