SI100B EE-Part MiniCPU

RISC-V Encoding

Yuxuan Li

VSPLab@SIST

2026.06.03

liyx22025@shanghaitech.edu.cn

目录

1 指令的编码：好的设计需要好的妥协

2 R 型指令

3 I 型指令

4 S/B 型指令

5 U/J 型指令

1 指令的编码：好的设计需要好的妥协

从汇编语言到机器语言

汇编语言便于人的阅读，然而，数字电路只能理解二进制。因此，汇编语言编写的程序必须要通过汇编器转换为用二进制表示的机器语言，才能在处理器上执行。

- 汇编语言：`add x3, x1, x2`
- 机器语言：`0000000 00010 00001 000 00011 0110011`

本节课的主要目标，就是研究指令编码，即汇编语言和机器语言之间的映射关系。

从汇编语言到机器语言

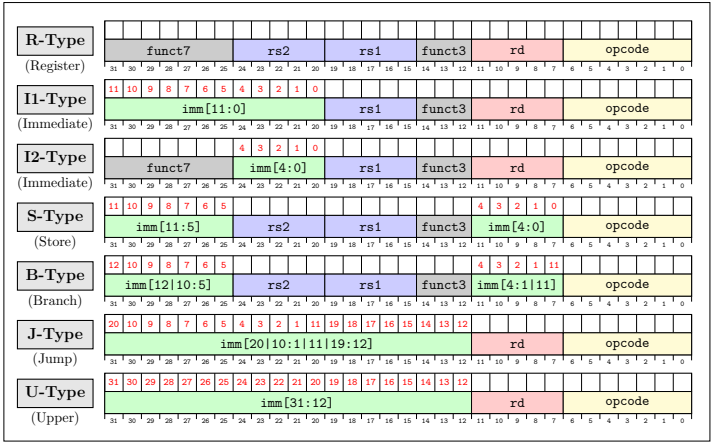
RISC-V 固定使用 32 位编码指令，部分字段编码指令类型，部分字段编码操作数

- `opcode = Instr[06:00]`，7 位：第一级的指令类型，所有指令均有该字段。
- `funct7 = Instr[31:25]`，7 位：第二级的指令类型，少数指令才有该字段。
- `funct3 = Instr[14:12]`，3 位：第三级的指令类型，多数指令都有该字段。
- `rs1 = Instr[19:15]`，5 位：指令的第一个源寄存器 `rs1` 的地址。
- `rs2 = Instr[24:20]`，5 位：指令的第二个源寄存器 `rs2` 的地址。
- `rd = Instr[11:07]`，5 位：指令的目标寄存器 `rd` 的地址。
- `imm`，编码和长度取决于指令类型：指令的立即数片段。

指令类型

RISC-V 有六种指令类型

- R 型：寄存器类型
- I 型：立即数类型
- S 型：存储类型
- B 型：分支类型
- J 型：跳转类型
- U 型：高位立即数类型



指令类型

- 指令类型所区分的，事实上就是立即数的编码方式！
- 指令的字段编码尽可能的在不同类型间保持了一致，以减轻指令译码的难度。
- 立即数的编码方式为何如此奇怪？（占空，对齐）
- 立即数扩展单元是处理器中一个重要模块：指令只能提供 5 位、12 位、20 位用于立即数的编码，需要按照不同类型的编码方式，将其扩展为 32 位的立即数。
- 指令集设计的关键，恰恰在于 32 位指令无法完整表示一个 32 位立即数。

2 R 型指令

R 型指令

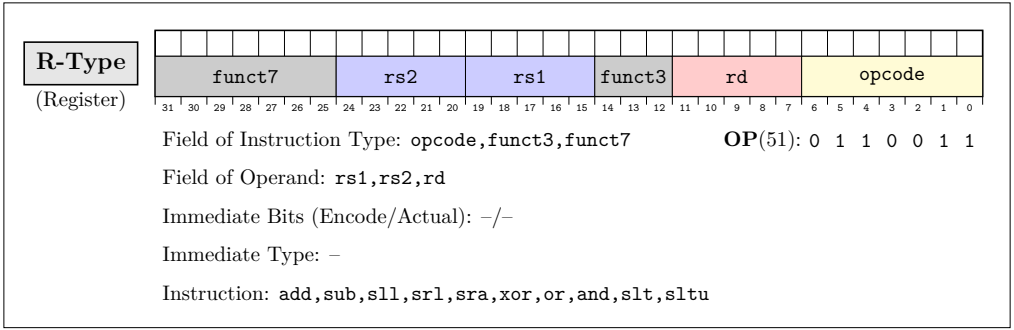

R-Type
(Register)
funct7
rs2
rs1
funct3
rd
opcode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Field of Instruction Type: opcode,funct3,funct7
OP(51): 0 1 1 0 0 1 1
Field of Operand: rs1,rs2,rd
Immediate Bits (Encode/Actual): –/–
Immediate Type: –
Instruction: add,sub,sll,srl,sra,xor,or,and,slt,sltu

图 1: R 型指令

R 型指令

R 型指令代表寄存器类型，其仅有 0110011 这一个 opcode 取值。指令译码时，首先根据 opcode 确定指令最宽泛的功能类型，例如 OP 的含义就是寄存器算术指令。

- 若 opcode = 0110011，记为 R/OP，代表这是寄存器算术指令。

R 型指令

R 型指令是少数具有 funct7 字段的指令类型

- 若 funct3 = 000，且 funct7 = 0000000，代表这是一条 add 指令。
- 若 funct3 = 000，且 funct7 = 0100000，代表这是一条 sub 指令。
- 若 funct3 = 110，代表这是一条 or 指令。
- 若 funct3 = 111，代表这是一条 and 指令。
- 若 funct3 = 010，代表这是一条 slt 指令。

R 型指令

由此可见，在 R/OP 中，funct3 字段指定具体的计算类型，例如加法、或、与、比较等，funct7 字段有 0000000 和 0100000 两个可能的取值，大部分情况下 funct7 都是取 0000000，但有时令 funct7 取 0100000 可以表达一个“变种指令”的含义。

- funct3 = 000，0000000 和 0100000 区分 add 和 sub（加法/减法）。
- funct3 = 101，0000000 和 0100000 区分 srl 和 sra（逻辑右移/算术右移）。

RISC-V 的 funct7 字段在编码上有非常大的冗余，这提供了充足的扩展空间。

表 1: R 型指令的列表（第一部分）

指令	用途	类型	opcode	f3	f7
`add rd,rs1,rs2`	`rd=rs1+rs2`	R	0110011	000	0000000
Add					寄存器加法
`sub rd,rs1,rs2`	`rd=rs1-rs2`	R	0110011	000	0100000
Sub					寄存器减法
`sll rd,rs1,rs2`	`rd=rs1<<rs2[4:0]`	R	0110011	001	0000000
Shift Left Logical					寄存器逻辑左移
`srl rd,rs1,rs2`	`rd=rs1>>rs2[4:0]`	R	0110011	101	0000000
Shift Right Logical					寄存器逻辑右移
`sra rd,rs1,rs2`	`rd=rs1>>>rs2[4:0]`	R	0110011	101	0100000
Shift Right Arithmetic					寄存器算数右移

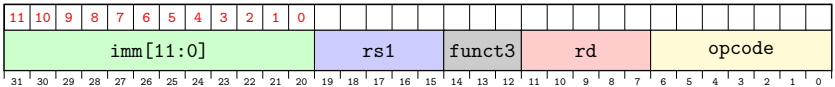
表 2: R 型指令的列表（第二部分）

指令	用途	类型	opcode	f3	f7
xor rd,rs1,rs2	rd=rs1^rs2	R	0110011	100	0000000
Xor					寄存器异或
or rd,rs1,rs2	rd=rs1|rs2	R	0110011	110	0000000
Or					寄存器或
and rd,rs1,rs2	rd=rs1&rs2	R	0110011	111	0000000
And					寄存器与
slt rd,rs1,rs2	rd=(rs1<rs2)	R	0110011	010	0000000
Set Less Than			寄存器当小于（有符号）时赋值		
sltu rd,rs1,rs2	rd=(rs1<rs2)	R	0110011	011	0000000
Set Less Than Unsigned			寄存器当小于（无符号）时赋值		

3 I 型指令

I1 型指令

I1-Type
(Immediate)



Field of Instruction Type: `opcode,funct3`

Field of Operand: `imm,rs1,rd`

Immediate Bits (Encode/Actual): 12/12

Immediate Type: Signed

Instruction: `addi,xori,ori,andi,slti,sltiu,lb,lh,lw,lbu,lhu,jalr`

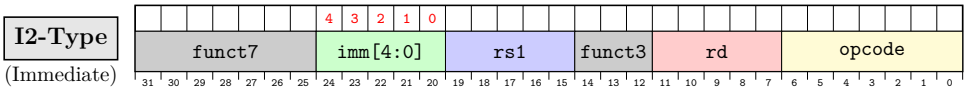
OP-IMM(19): 0 0 1 0 0 1 1

LOAD(3): 0 0 0 0 0 1 1

JALR(103): 1 1 0 0 1 1 1

图 2: I1 型指令

I2 型指令



Field of Instruction Type: `opcode,funct3,funct7` **OP-IMM**(19): 0 0 1 0 0 1 1

Field of Operand: `imm,rs1,rd`

Immediate Bits (Encode/Actual): 05/05

Immediate Type: Unsigned

Instruction: `slli,srli,srai`

图 3: I2 型指令

I 型指令

I 型指令代表立即数类型，其包含了三个 opcode 取值

- 若 opcode = 0010011，记为 I/OP-IMM，代表这是立即数算术指令。
- 若 opcode = 0000011，记为 I/LOAD，代表这是读取指令。
- 若 opcode = 1100111，记为 I/JALR，代表这是 jalr 指令。

I 型指令

I 型指令不仅包含 `addi` 这样的立即数算术指令，也有 `lw` 这样的读取指令。

- I 型指令的本质，是对 `rs1` 和 `imm` 做某种运算，并向 `rd` 写入某个结果。
- 对于 `addi`：计算 `rs1 + imm`，写入 `rd = rs1 + imm`。
- 对于 `lw` ：计算 `rs1 + imm`，写入 `rd = mem[rs1 + imm]`。
- 虽然两者写回 `rd` 的数据来源不同，但是完全可以共享相同的指令编码！

I 型指令

I 型指令使用 12 位来编码立即数 `imm[11:0]`，然而，参与计算的立即数必须要先被扩展到 32 位，因此，按何种原则填充高位就是一个值得关注的问题。RISC-V 指令几乎都采用有符号扩展，即用原先的最高位填充高位，这确保了负数在扩展后仍然是相同的负数。也因此，汇编中才可以使用 `add x3, x1, -8` 这样的负立即数。

立即数扩展

二进制数的扩展有两种方式

- 无符号扩展（Zero Extend），将高位用零填充。
- 有符号扩展（Sign Extend），将高位用原先的最高位填充。

- `ZE(0000 0000 0001)= 0000 0000 0000 0000 0000 0000 0000 0001`
- `ZE(1111 1111 1111)= 0000 0000 0000 0000 0000 1111 1111 1111`

- `SE(0000 0000 0001)= 0000 0000 0000 0000 0000 0000 0000 0001`
- `SE(1111 1111 1111)= 1111 1111 1111 1111 1111 1111 1111 1111`

I 型指令

I 型指令的立即数占用了原本 `funct7` 的空间，仅有 `funct3` 可用

- 对于 I/OP-IMM，`funct3` 的含义与原先 R/OP 完全相同，这在一定程度上复用译码，而对于需要 `funct7` 介入区分的 `add/sub` 和 `srl/sra`，前者由于并不存在 `subi` 指令而解决，后者属于特殊的 I2 型指令，其仍然会有 `funct7` 字段。
- 对于 I/LOAD，`funct3` 代表读取的数据大小，对于 `lw` 为 `010`。
- 对于 I/JALR，`funct3` 只有 `000` 一个取值，对应该类型唯一的 `jalr` 指令。

I 型指令

I 型指令分为 I1 和 I2 两个亚型，I1 型就是常规的 I 型指令，I2 型是针对三条立即数位移指令 `slli`、`srli`、`srai` 设置的。对于一个 32 位数，其有效的左右移位数的范围就是 0 至 31，因此，用于表示左右移位数的立即数 `imm[4:0]` 只需要 5 位来编码就足够了，空余的位置交还给 `funct7`，以和 R/OP 相同的方式区分 `srli` 和 `srai`。

I2 型指令是唯一需要进行无符号扩展的特例，但实际上，由于计算左移和右移时通常会截断低 5 位，因此，这里采用无符号扩展还是有符号扩展事实上是无关紧要的。

I 型指令

I/LOAD 中的 `funct3`：最高位代表“有符号/无符号”，低两位代表“数据宽度”

- 若 `funct3 = 000`，代表这是 `lb` 指令，读取 8 位的字节（Byte）。
- 若 `funct3 = 001`，代表这是 `lh` 指令，读取 16 位的半字（Half）。
- 若 `funct3 = 010`，代表这是 `lw` 指令，读取 32 位的字（Word）。
- 若 `funct3 = 100`，代表这是 `lbu` 指令。
- 若 `funct3 = 101`，代表这是 `lhu` 指令。

指令 `lbu` 和 `lhu` 相较 `lb` 和 `lbu` 的区别是：读取的 8 位和 16 位的数据片段，是以无符号方式还是有符号方式扩展到 32 位？因此，并不需要一条所谓的 `lwu` 指令。

表 3: I 型指令的列表（第一部分）

指令	用途	类型	opcode	f3	f7
addi rd,rs1,imm	rd=rs1+SE(imm)	I1	0010011	000	-------
Add Immediate					立即数加法
slli rd,rs1,imm	rd=rs1<<imm	I2	0010011	001	0000000
Shift Left Logical Immediate					立即数逻辑左移
srli rd,rs1,imm	rd=rs1>>imm	I2	0010011	101	0000000
Shift Right Logical Immediate					立即数逻辑右移
srai rd,rs1,imm	rd=rs1>>>imm	I2	0010011	101	0100000
Shift Right Arithmetic Immediate					立即数算数右移
jalr rd,rs1,imm	rd=pc+4, pc=rs1+SE(imm)	I1	1100111	000	-------
Jump and Link Register					跳转至寄存器并链接

表 4: I 型指令的列表（第二部分）

指令	用途	类型	opcode	f3	f7
xori rd,rs1,imm	rd=rs1^SE(imm)	I1	0010011	100	-------
Xor Immediate					立即数异或
ori rd,rs1,imm	rd=rs1|SE(imm)	I1	0010011	110	-------
Or Immediate					立即数或
andi rd,rs1,imm	rd=rs1&SE(imm)	I1	0010011	111	-------
And Immediate					立即数与
slti rd,rs1,imm	rd=(rs1<SE(imm))	I1	0010011	010	-------
Set Less Than Immediate					立即数当小于（有符号）时赋值
sltiu rd,rs1,imm	rd=(rs1<SE(imm))	I1	0010011	011	-------
Set Less Than Immediate Unsigned					立即数当小于（无符号）时赋值

表 5: I 型指令的列表（第三部分）

指令	用途	类型	opcode	f3	f7
lb rd,imm(rs1)	rd=SE([rs1+SE(imm)][7:0])	I1	0000011	000	-------
Load Byte					读取字节
lh rd,imm(rs1)	rd=SE([rs1+SE(imm)][15:0])	I1	0000011	001	-------
Load Half					读取半字
lw rd,imm(rs1)	**rd=[rs1+SE(imm)]**	**I1**	**0000011**	**010**	**-------**
Load Word					**读取字**
lbu rd,imm(rs1)	rd=ZE([rs1+SE(imm)][7:0])	I1	0000011	100	-------
Load Byte Unsigned					读取无符号字节
lhu rd,imm(rs1)	rd=ZE([rs1+SE(imm)][15:0])	I1	0000011	101	-------
Load Half Unsigned					读取无符号半字

4 S/B 型指令

S 型指令

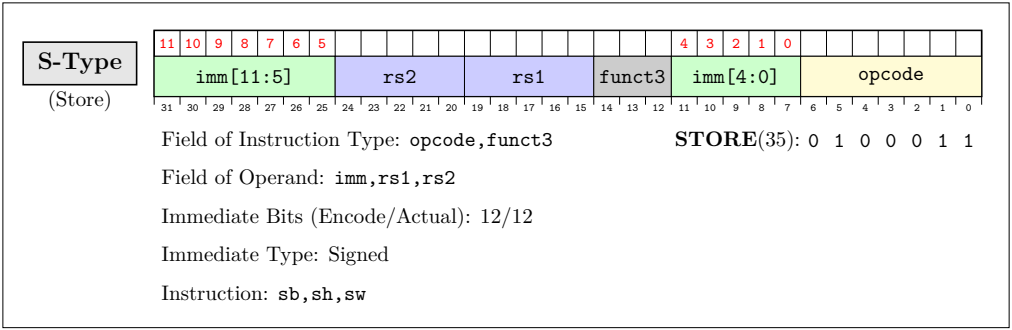
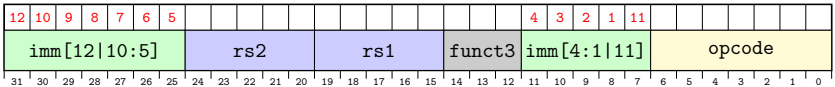

S-Type
(Store)
11 10 9 8 7 6 5
4 3 2 1 0
imm[11:5]
rs2
rs1
funct3
imm[4:0]
opcode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Field of Instruction Type: opcode,funct3
STORE(35): 0 1 0 0 0 1 1
Field of Operand: imm,rs1,rs2
Immediate Bits (Encode/Actual): 12/12
Immediate Type: Signed
Instruction: sb,sh,sw

图 4: S 型指令

B 型指令

B-Type
(Branch)



Field of Instruction Type: `opcode,funct3`

BRANCH(99): 1 1 0 0 0 1 1

Field of Operand: `imm,rs1,rs2`

Immediate Bits (Encode/Actual): 12/13

Immediate Type: Signed

Instruction: `beq,bne,blt,bge,bltu,bgeu`

图 5: B 型指令

S/B 型指令

S 型指令代表存储类型，B 型指令代表分支类型

- 若 opcode = 0100011，记为 S/STORE，代表这是一条存储指令。
- 若 opcode = 1100011，记为 B/BRANCH，代表这是一条分支指令。

S/B 型指令

S/B 型指令一起讨论的原因是两者的编码很接近，仅在 12 位立即数编码上有差异

- 在 S 型指令中，12 位编码的是立即数的 `imm[11:0]`。
- 在 B 型指令中，12 位编码的是立即数的 `imm[12:1]`。

因此，对于 B 型指令，要先“填充低位”，用零填充 `imm[12:1]` 未指定的低位，得到 `imm[12:0]`，换言之，B 型指令用 12 位的编码表示了一个 13 位的立即数，只不过这个立即数的最低位固定为零罢了。随后再“扩展高位”，有符号扩展至 32 位。

S/B 型指令

B 型指令的立即数设定为 2 的倍数的原因在于：指令每条 32 位占用 4 个字节，在进行跳转时，指令地址的偏移至少都是 4 的倍数。由于 RV32C 的压缩扩展中会出现 16 位占用 2 个字节的指令，因此这里立即数设定为 2 的倍数而不是 4 的倍数。

S 型指令不能采用类似设定，因为会涉及到 `lb` 和 `lh` 这样的字节和半字访问。

S/B 型指令

B/BRANCH 中的 `funct3` 代表了分支指令的类型

- 若 `funct3 = 000`，代表这是 `beq` 指令，跳转条件是“相等”。
- 若 `funct3 = 001`，代表这是 `bne` 指令，跳转条件是“不等”。
- 若 `funct3 = 100`，代表这是 `blt` 指令。跳转条件是“有符号小于”。
- 若 `funct3 = 110`，代表这是 `bge` 指令。跳转条件是“有符号大等”。
- 若 `funct3 = 101`，代表这是 `bltu` 指令。跳转条件是“无符号小于”。
- 若 `funct3 = 111`，代表这是 `bgeu` 指令。跳转条件是“无符号大等”。
- 比较运算是补码机制下，极少数需要区分有符号和无符号的运算。

S/B 型指令

S/STORE 中的 `funct3` 和 I/LOAD 中的 `funct3` 功能基本一致

- 若 `funct3 = 000`，代表这是 `sb` 指令，写入 8 位的字节（Byte）。
- 若 `funct3 = 001`，代表这是 `sh` 指令，写入 16 位的半字（Half）。
- 若 `funct3 = 010`，代表这是 `sw` 指令，写入 32 位的字（Word）。

请注意，并不存在 `sbu` 指令和 `shu` 指令，写入时并不需要扩展到 32 位。

表 6: S/B 型指令的列表（第一部分）

指令	用途	类型	opcode	f3	f7
sb rs2,imm(rs1) Store Byte	[rs1+SE(imm)][7:0]=rs2[7:0]	S	0100011	000	------- 存储字节
sh rs2,imm(rs1) Store Half	[rs1+SE(imm)][15:0]=rs2[15:0]	S	0100011	001	------- 存储半字
sw rs2,imm(rs1) Store Word	[rs1+SE(imm)]=rs2	S	0100011	010	------- 存储字
beq rs1,rs2,lab Branch If Equal	if(rs1==rs2) pc+=SE({imm,1'b0})	B	1100011	000	------- 若相等则跳转
bne rs1,rs2,lab Branch If Not Equal	if(rs1!=rs2) pc+=SE({imm,1'b0})	B	1100011	001	------- 若不等则跳转

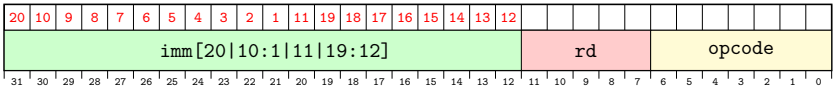
表 7: S/B 型指令的列表（第二部分）

指令	用途	类型	opcode	f3	f7
`blt rs1,rs2,lab`	`if(rs1< rs2) pc+=SE({imm,1'b0})`	B	1100011	100	-------
Branch If Less Than				若小于（有符号）则跳转	
`bltu rs1,rs2,lab`	`if(rs1< rs2) pc+=SE({imm,1'b0})`	B	1100011	110	-------
Branch If Less Than Unsigned				若小于（无符号）则跳转	
`bge rs1,rs2,lab`	`if(rs1>=rs2) pc+=SE({imm,1'b0})`	B	1100011	101	-------
Branch If Greater Than or Equal				若大等（有符号）则跳转	
`bgeu rs1,rs2,lab`	`if(rs1>=rs2) pc+=SE({imm,1'b0})`	B	1100011	111	-------
Branch If Greater Than or Equal Unsigned				若大等（无符号）则跳转	

5 U/J 型指令

J 型指令

J-Type
(Jump)



Field of Instruction Type: `opcode`

JAL(111): 1 1 0 1 1 1 1

Field of Operand: `imm,rd`

Immediate Bits (Encode/Actual): 20/21

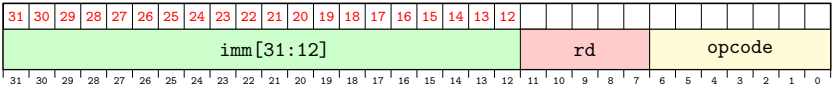
Immediate Type: Signed

Instruction: `jal`

图 6: J 型指令

U 型指令

U-Type
(Upper)



Field of Instruction Type: `opcode`

Field of Operand: `imm,rd`

Immediate Bits (Encode/Actual): 20/32

Immediate Type: –

Instruction: `lui,auipc`

LUI(55): 0 1 1 0 1 1 1

AUIPC(23): 0 0 1 0 1 1 1

图 7: U 型指令

U/J 型指令

J 型指令代表跳转类型，U 型指令代表高位立即数类型

- 若 `opcode = 1101111`，记为 J/JAL，代表这是一条 `jal` 指令。
- 若 `opcode = 0110111`，记为 U/LUI，代表这是一条 `lui` 指令。
- 若 `opcode = 0010111`，记为 U/AUIPC，代表这是一条 `auipc` 指令。

U/J 型指令

U/J 型指令均采用了 20 位立即数编码，只有 opcode 字段可以用于区分指令

- 在 J 型指令中，20 位编码的是立即数的 imm[20:1]。
- 在 U 型指令中，20 位编码的是立即数的 imm[31:20]。

J 型指令的立即数和 B 型指令的立即数的风格是非常相似的，均利用了指令地址至少是 2 的倍数的特性。需要注意的是，jal 是唯一的 J 型指令，jalr 是 I 型指令。

U 型指令基本和高位立即数有关。例如 addi 只能加载一个 12 位的立即数，如果要将一个很大的数加载到寄存器中，就可以先用 lui 加载高 20 位，随后再用 addi 加载低 12 位。不过，由于 addi 采用有符号扩展，在某些边界情况下会有额外麻烦。

表 8: U/J 型指令的列表（第一部分）

指令	用途	类型	opcode	f3	f7
`jal rd,lab`	`rd=pc+4, pc+=SE({imm,1'b0})`	J	1101111	---	-------
Jump and Link					跳转并链接
`lui rd,imm`	`rd={imm,12'b0}`	U	0110111	---	-------
Load Upper Immediate					加载高位立即数
`auipc rd,imm`	`rd=pc+{imm,12'b0}`	U	0010111	---	-------
Add Upper Immediate to PC					将当前地址与高位立即数相加

谢谢大家！