INTRODUCTION TO RISC-V

RISC-V入门教程

ISA | 汇编指令 | 系统编程 | 组成原理 | 嵌入式应用

数据表示

主讲 邢建国

中国开放指令生态（RISC-V）联盟 | 浙江中心
China RICS-V Alliance | Zhejiang Center

浙江图灵算力研究院
ZHEJIANG TURING INSTITUTE

RISC-V®

数据表示

- 无符号、有符号数
- 浮点数IEEE 754
- 文本表示
- 内存中的数据和指令

无符号数与有符号数

- 计算机中基本存储单元
  - 1位二进制：可以表示$2^1=2$种编码
  - 2位二进制：可以表示$2^2=4$种编码
  - 3位二进制：可以表示$2^3=8$种编码
  - …
  - n位二进制：可以表示2^n种编码

无符号数与有符号数

3位编码	无符号数
000	0_{10}
001	1_{10}
010	2_{10}
011	3_{10}
100	4_{10}
101	5_{10}
110	6_{10}
111	7_{10}

无符号数与有符号数

3位编码	值	
	无符号数	有符号数
000	0_{10}	0_{10}
001	1_{10}	1_{10}
010	2_{10}	2_{10}
011	3_{10}	3_{10}
100	4_{10}	-0_{10}
101	5_{10}	-1_{10}
110	6_{10}	-2_{10}
111	7_{10}	-3_{10}

无符号数与有符号数

3位编码	值			
	无符号数	有符号数	1补码	2补码
000	0_{10}	0_{10}	0_{10}	0_{10}
001	1_{10}	1_{10}	1_{10}	1_{10}
010	2_{10}	2_{10}	2_{10}	2_{10}
011	3_{10}	3_{10}	3_{10}	3_{10}
100	4_{10}	-0_{10}	-3_{10}	-4_{10}
101	5_{10}	-1_{10}	-2_{10}	-3_{10}
110	6_{10}	-2_{10}	-1_{10}	-2_{10}
111	7_{10}	-3_{10}	-0_{10}	-1_{10}

补码表示

- RISC-V使用2补码来表示整数
  - n位补码表示范围为：【-2^{n-1}，$+2^{n-1}-1$】
  - 正数的补码和二进制编码相同，符号位为0
  - 负数的补码：其绝对值二进制编码按位取反，加上1

- 补码运算的好处在于减法可以用补码加法来实现
- $(A+B)_{补}=A_{补}+B_{补}$
- $(A-B)_{补}=A_{补}+(-B)_{补}$

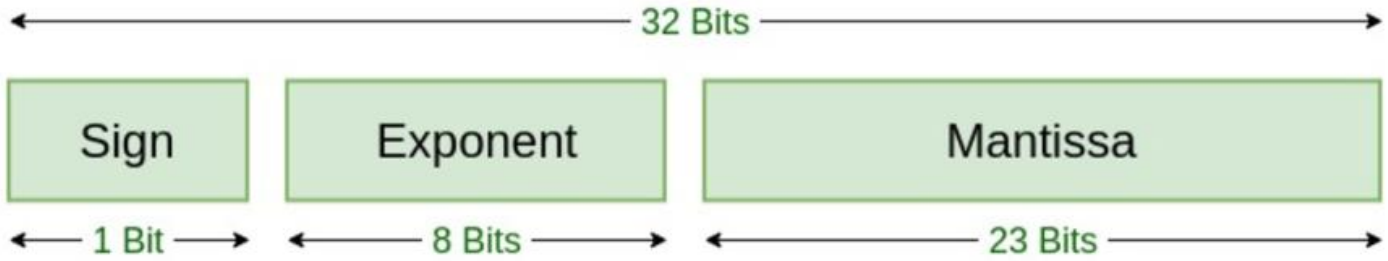
	无符号数	补码表示
	010_2	2_{10}
$+$	100_2	$+4_{10}$
	110_2	6_{10}

	无符号数	补码表示
	2_{10}	2_{10}
$+$	4_{10}	$+-4_{10}$
	6_{10}	-2_{10}

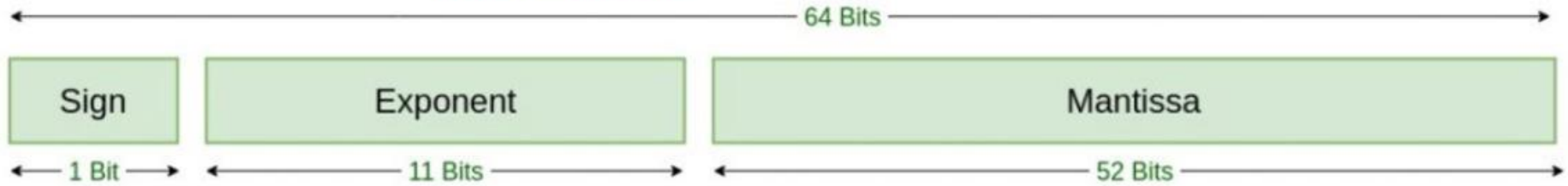
■ 浮点数：IEEE 754标准

➢ RISC-V支持IEEE754-2019

单精度float

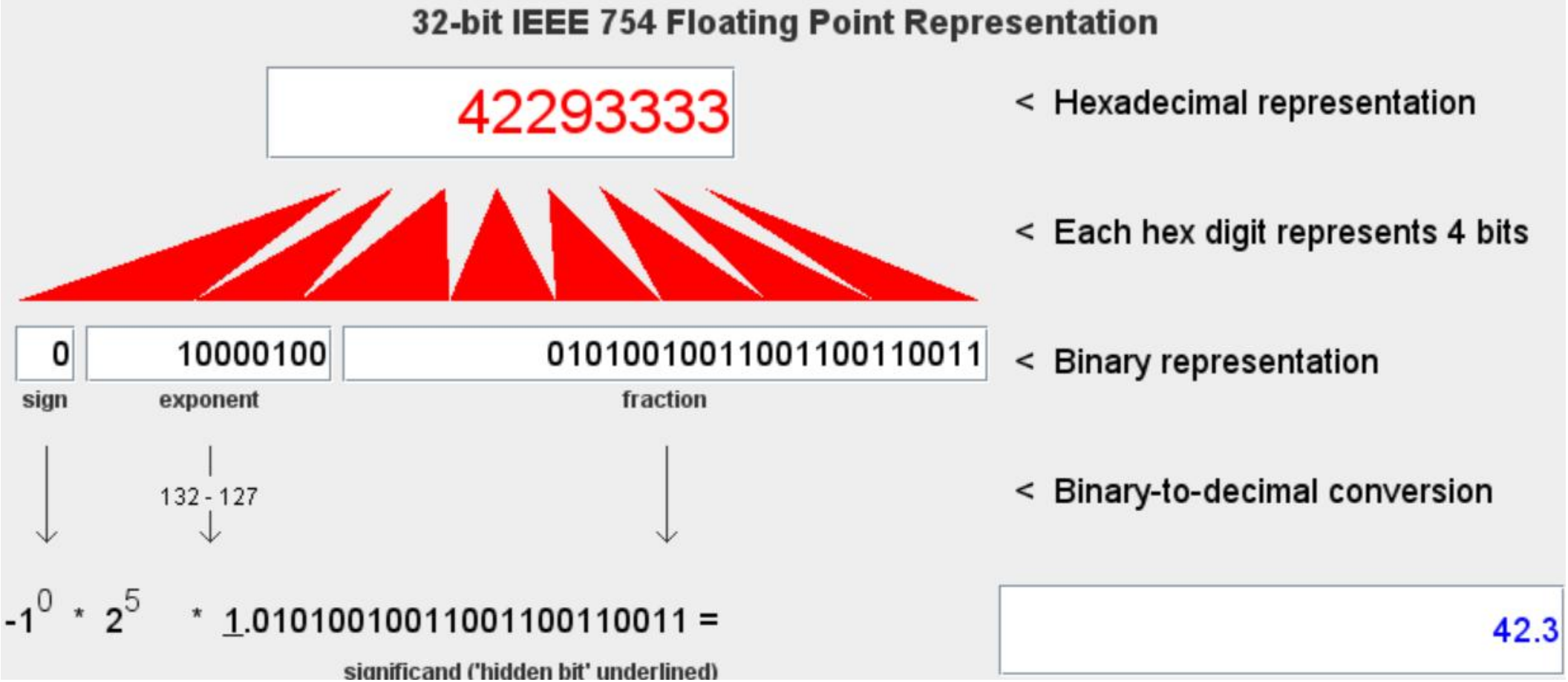


双精度double



BIAS
127
1023

■ 浮点数：IEEE 754标准



■ 浮点数：IEEE 754标准

EXPONENT	MANTISA	VALUE
0	0	exact 0
255	0	Infinity
0	not 0	denormalised
255	not 0	Not a number (NAN)

文本符号的表示

- ASCII
  - 7位
- Extended ASCII
  - 8位

Binary	Hex.	Dec.	Char.
...			
0100001_2	21_{16}	33_{10}	!
0100010_2	22_{16}	34_{10}	"
...			
0101100_2	$2C_{16}$	44_{10}	,
0101101_2	$2D_{16}$	45_{10}	-
0101110_2	$2E_{16}$	46_{10}	.
0101111_2	$2F_{16}$	47_{10}	/
0110000_2	30_{16}	48_{10}	0
0110001_2	31_{16}	49_{10}	1
0110010_2	32_{16}	50_{10}	2
...			
0111000_2	38_{16}	56_{10}	8
0111001_2	39_{16}	57_{10}	9
...			

Binary	Hex.	Dec.	Char.
1000001_2	41_{16}	65_{10}	A
1000010_2	42_{16}	66_{10}	B
...			
1011001_2	59_{16}	89_{10}	Y
1011010_2	$5A_{16}$	90_{10}	Z
...			
1100001_2	61_{16}	97_{10}	a
1100010_2	62_{16}	98_{10}	b
...			
1111001_2	79_{16}	121_{10}	y
1111010_2	$7A_{16}$	122_{10}	z
...			
1111100_2	$7C_{16}$	124_{10}	|
1111101_2	$7D_{16}$	125_{10}	}
1111110_2	$7E_{16}$	126_{10}	~

文本符号的表示：Unicode

- Unicode字符集
  - Universal Coded Character Set
  - 包含了世界上大多数语言的字符，以及各种符号和表情。
- Unicode的UTF-8编码
  - UTF-8兼容ASCII编码，ASCII文本都可以被视为UTF-8文本，无需任何转换
  - UTF-8使用1到4个字节来表示一个字符，这取决于字符的Unicode码点。
  - 对于0到127的码点（即ASCII字符集），UTF-8使用一个字节，与ASCII编码相同。
  - 对于128到2047的码点，UTF-8使用两个字节。
  - 对于2048到65535的码点，UTF-8使用三个字节。
  - 对于65536以上的码点，UTF-8使用四个字节。

UTF-8

字节数	编码位数	UTF-8字节流
1	7	0xxxxxxx
2	11	110xxxxx 10xxxxxx
3	16	1110xxxx 10xxxxxx 10xxxxxx
4	21	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
5	26	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
6	31	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
7	36	11111110 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
8	42	11111111 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Unicode

hello 世界

hello \u4e16\u754c

UTF-8

hello 世界

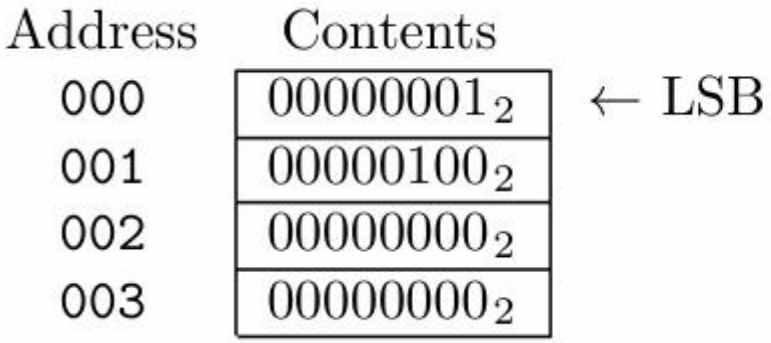
内存中的字符串

```c
#include<stdio.h>
char name1[] = "John";
char name2[] = {0x4a, 0x6f, 0x68, 0x6e, 0x00};
int main()
{
  printf("Name 1: \"%s\"\n", name1);
  printf("Name 2: \"%s\"\n", name2);
  printf("Size of name 1 = %d\n", sizeof(name1));
  printf("Size of name 2 = %d\n", sizeof(name2));
  return 0;
}
```

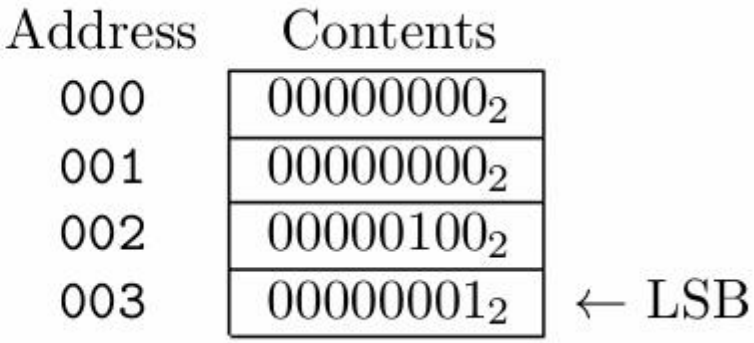
```
Name 1: "John"
Name 2: "John"
Size of name 1 = 5
Size of name 2 = 5
```

内存中的数

32-bit number $00000000\,00000000\,00000100\,00000001_2$ (1025_{10})



小端方式 (little-endian)

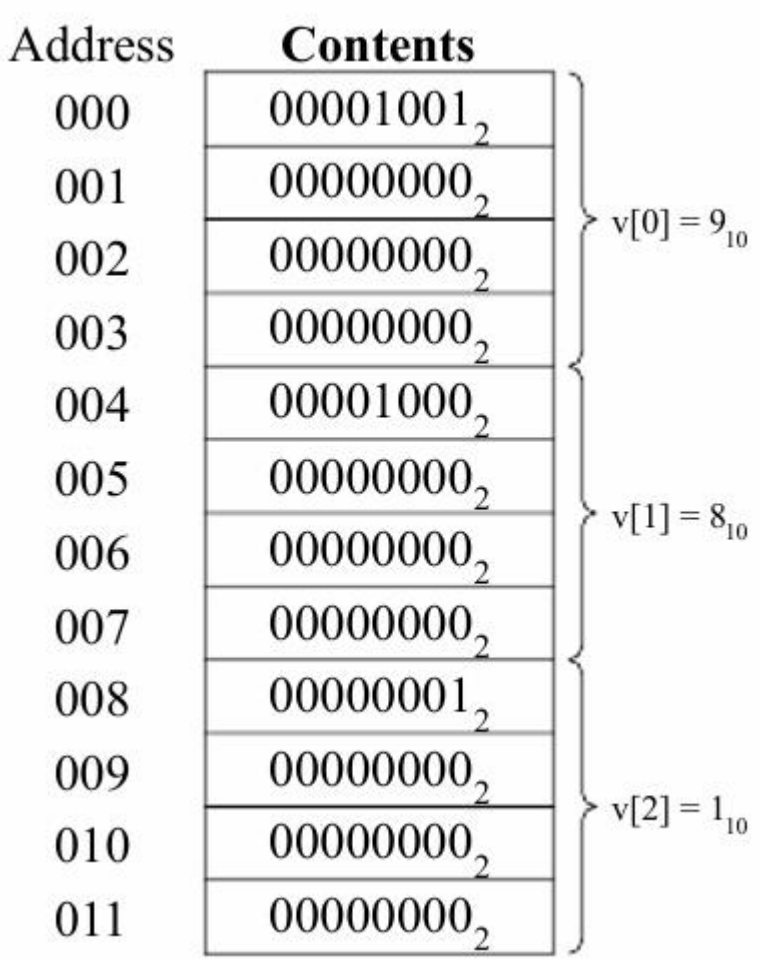


大端方式 (big-endian)

RISC-V采用小端方式

内存中的数组

```
1 int V[] = {9, 8, 1};
2 void print_V()
3 {
4     printf("First element = %d\n", V[0]);
5     printf("Last  element = %d\n", V[2]);
6 }
```



小端方式（little-endian）

二维数组

```
1 int M[][] = { {7, 9, 11},
2               {2, 8, 1} };
3 void print_M()
4 {
5     printf("Element M[0][0] = %d\n", M[0][0]);
6     printf("Element M[1][2] = %d\n", M[1][2]);
7 }
```

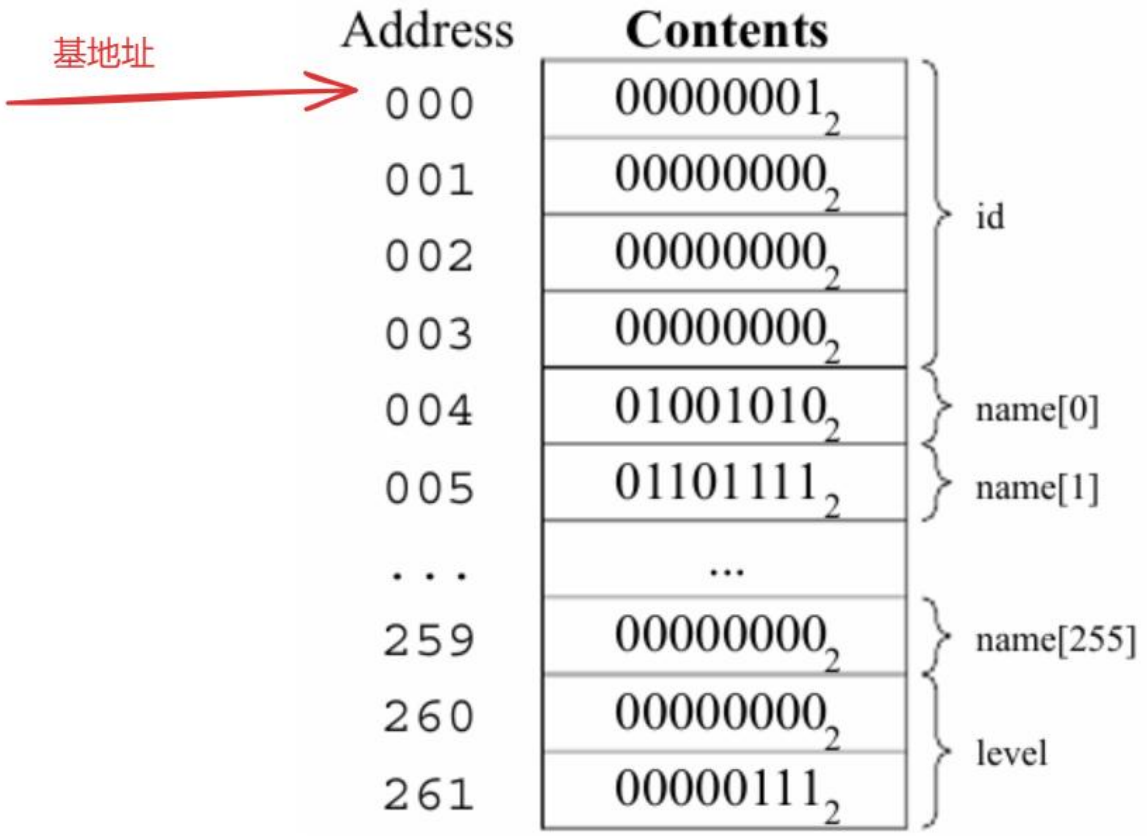
二维数组在内存中按行排列（C、C++、JAVA、MATLAB等）

$$\&A[x][y] = A_{addr} + \underbrace{x \times elem_{size} \times N}_{\text{offset 1}} + \underbrace{y \times elem_{size}}_{\text{offset 2}}.$$

Fortran、Matlab、R等在内存中按列排列

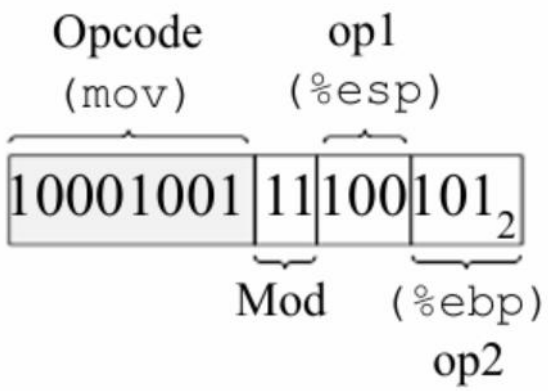
内存中的结构体

```
1 struct user_id {
2         int   id;
3         char  name[256];
4     short level;
5 };
6
7 struct user_id manager;
8
```



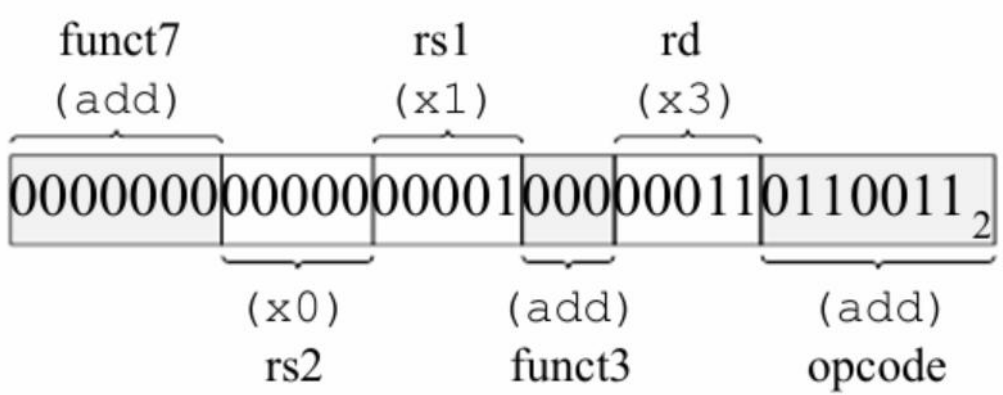
小端方式（little-endian）

内存中的指令




```
mov %esp, %ebp
```

x86汇编指令

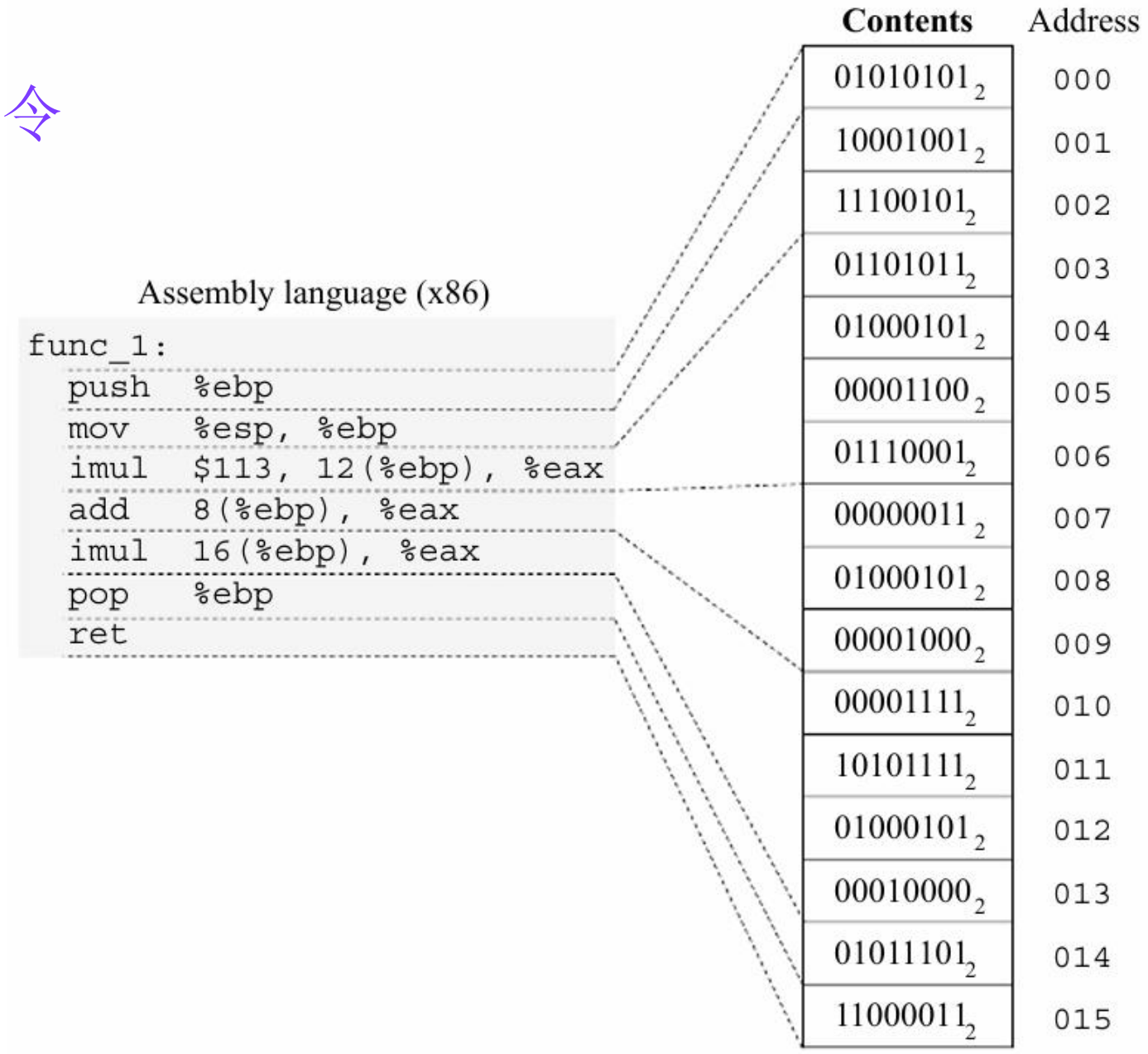



```
add x3, x1, x0
```

RISC-V汇编指令

小端方式 (little-endian)

内存中的指令


Assembly language (x86)
func_1:
push %ebp
mov %esp, %ebp
imul $113, 12(%ebp), %eax
add 8(%ebp), %eax
imul 16(%ebp), %eax
pop %ebp
ret
Contents
Address
01010101 2
10001001 2
11100101 2
01101011 2
01000101 2
00001100 2
01110001 2
00000011 2
01000101 2
00001000 2
00001111 2
10101111 2
01000101 2
00010000 2
01011101 2
11000011 2
000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015

RISC-V®

数据表示

- 无符号、有符号数
- 浮点数IEEE 754
- 文本表示
- 内存中的数据和指令