RISC-V ISA

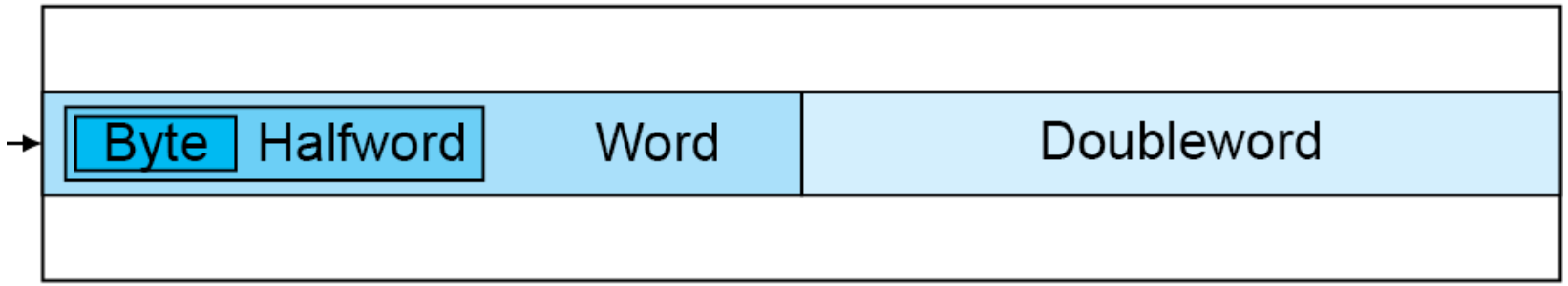
Little vs Big Endian
Sign vs Zero Extension

Sparsh Mittal

Little- and big-endian

(This will be useful for understanding load-store instructions in RISC-V)

Units of memory



Units of memory

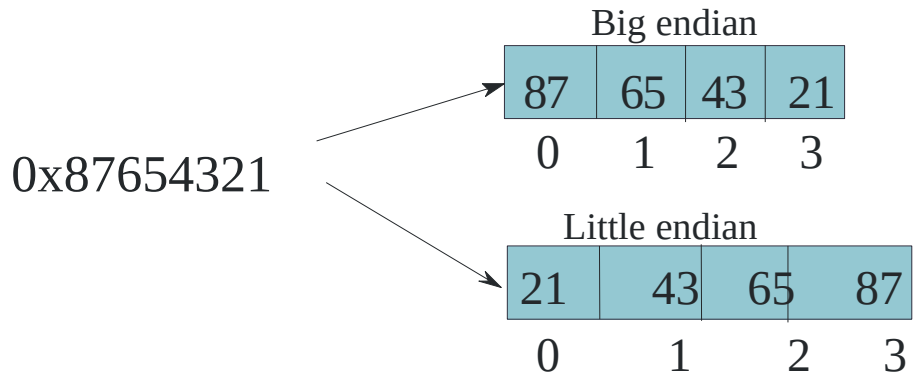
- Byte = 8b
- Halfword = 16b
- Word = 32b
- Doubleword = 64b

Datatypes in C

- char = 8b
- short = 16b
- int = 32b
- Long int or double = 64b

Little vs Big Endian

- How are multibyte variables stored in memory ?
  - Example : How is a 4 byte integer stored ?
  - Save the 4 bytes in consecutive locations
  - Little endian representation (used in ARM, RISC-V and x86) → The LSB is stored in the lowest location
  - Big endian representation (Sun Sparc, IBM PPC) → The MSB is stored in the lowest location



- Note the order of the storage of bytes

Solved Question

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00			
90000001	00	sum	int (4 bytes)	000000FF(255_{10})
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Is it big-endian or little-endian? Answer: big-endian

The first value (sum) is 000000FF. Here, the least-significant byte is FF, which is stored at highest location, i.e., address 90000003.

Little Endian

- RISC-V chose little-endian byte ordering because it is dominant commercially.
- All x86-32 systems, and Apple iOS, Google Android OS, and Microsoft Windows for ARM are all little endian.

Understanding sign/zero extension

Extension

- Extension is required when we want to preserve the numeric value, while we represent a number using more bits; or store it in a register with more number of bits.

- There are two possibilities: sign-extension and zero-extension
- Sign-extension: replicate the sign bit to the left
- Zero-extension: replicate zero bit to the left.

Understanding extension

Case 1: Assume our 8b data is 1101 0011. We need to store it in a 16b register.

Sign- extension → 1111 1111 1101 0011

Zero- extension → 0000 0000 1101 0011

Case 2: Assume our 8b data is 0101 0011. We need to store it in a 16b register.

Sign- extension → 0000 0000 0101 0011

Zero- extension → 0000 0000 0101 0011

Notice that in case 2, sign- and zero-extension have the same impact.

Sign extension example.

- Write the value of -7 in 32 bits.

  $(-7)_{10} = (1001)_2$ # 2's complement representation.

- Sign extension in 32-bit representation:

  $(-7)_{10} = (\mathbf{1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001})_2$