2.3 Memory alignment and endianness

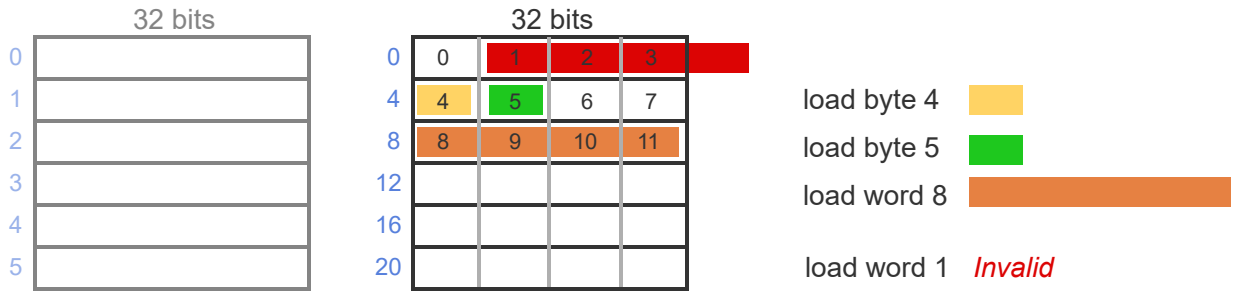
Memory alignment

A particular memory may store a sequence of 32-bit wide words (instructions or data). One might assume addresses for each word would increment by 1, as in: 0, 1, 2, 3, 4, 5, etc. However, each byte in a word can be addressed individually. Thus, addresses of each word increment by 4: 0, 4, 8, 12, 16, etc. ***Memory alignment*** is the restriction of word addresses to multiples of 4 (or other multiples for different processors).

Instructions that load or store words must use addresses that are multiples of four. Instructions that load or store bytes may use any address.

PARTICIPATION ACTIVITY | 2.3.1: Memory alignment: Because each byte is addressable, word addresses are multiples of 4.

1 2 3 4 5 2x speed



Word instructions (load or store) must be aligned: Multiples of 4 only.

Captions

1. One might assume that word addresses increment by 1.
2. However, each byte is addressable.
3. Thus, word addresses increment by 4.
4. Byte instructions refer only to the indicated byte. Word instructions refer to the entire word.
5. Word instructions (load or store) must be aligned: Multiples of 4 only.

Feedback?

PARTICIPATION ACTIVITY 2.3.2: Memory alignment.

Consider the above animation on memory alignment.

1) How many bytes exist per word?

4

Check Show answer

Correct

4

Each word is 32 bits. A byte is 8 bits. Thus 32 / 8 = 4 bytes exist per word.

2) How many byte addresses exist for one word?

4

Check Show answer

Correct

4

Each word has 4 bytes, and each byte needs an address. Thus, 4 addresses exist per word.

3) What are the byte addresses for the bytes in the word starting at address 12? Type as: 0, 1, 2, 3

12, 13, 14, 15

Check Show answer

Correct

12, 13, 14, 15

That word's first byte has address 12, the second byte 13, the third 14, and the fourth 15.

4) Is storing a byte into address 15 allowed? Type yes or no.

yes

Check Show answer

Correct

yes

Each byte in memory has an address, so no memory alignment restrictions exist for byte addresses.

5) Is storing a word into address 15 allowed? Type yes or no.

no

Check Show answer

Correct

no

Word addresses must be multiples of 4, but 15 is not a multiple of 4. Words exist at address 12 and at 16. Storing a word, which is 4 bytes, into 15 would require putting one byte into word 12's last byte, and the other three bytes into word 16's first three bytes. Such a mess is not supported.

6) A programmer wishes to load the last word of the memory shown in the animation. What address should be used in the load word instruction?

20

Check Show answer

Correct

20

The last word's address is 20. Loading word 20 will thus load the four bytes of that word. Note that word 20's 4 bytes have addresses 20, 21, 22, and 23, but those addresses are only relevant when loading a particular byte, not when loading an entire word.

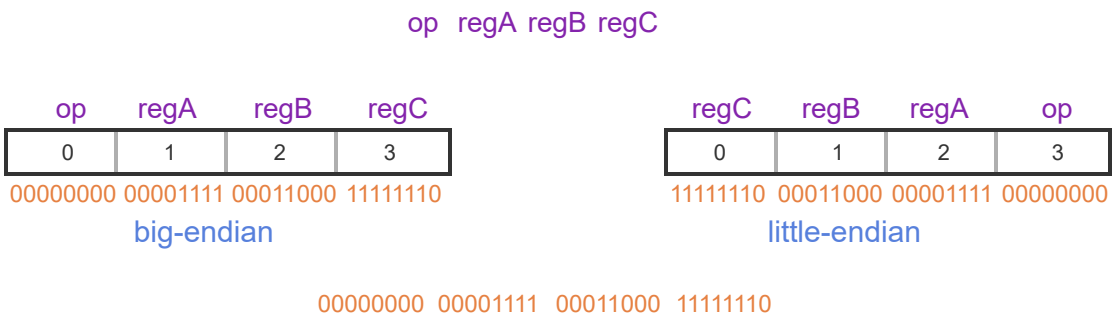
Feedback?

Endianness

Endianness refers to whether bytes in a word are ordered starting with the most-significant byte first (***big-endian***) or the least-significant byte first (***little-endian***). Some processors use big-endian format, others use little-endian.

PARTICIPATION ACTIVITY 2.3.3: Big-endian vs. little-endian.

1 2 3 4 2x speed



Captions

1. An instruction may require one byte for op, one for regA, one for regB, and one for regC (not real; for example purposes only).
2. Big endian stores the "most-significant" byte (op) first.
3. In contrast, little-endian stores the least-significant byte first.
4. Similarly for binary numbers.

Feedback?

Endianness only impacts the ordering of bytes; the bits within the byte remain in the same order. Ex: 00001111 remains 00001111 for either big or little endian formats, and does not become 11110000.

Programmers usually need not be concerned with endianness, unless doing byte-level operations within a word (which is rare).

PARTICIPATION ACTIVITY 2.3.4: Big-endian.

The binary number 00000000 00001111 11111111 11000000 (1048512 in decimal) is to be stored in word 20 in big-endian format. Indicate the byte address of each byte.

If unable to drag and drop, refresh the page.

00000000	20 The most-significant byte (00000000) will be stored in the first byte address.	**Correct**
00001111	21 The second byte is stored in this second byte address.	**Correct**
11111111	22 The third byte is stored in this third byte address.	**Correct**
11000000	23 The least-significant byte (11000000) will be stored in the last byte address.	**Correct**

Reset

Feedback?

PARTICIPATION ACTIVITY 2.3.5: Little-endian.

The binary number 00000000 00001111 11111111 11000000 (1048512 in decimal) is to be stored in word 20 in little-endian format. Indicate the byte address of each byte.

If unable to drag and drop, refresh the page.

11000000	20 The least-significant byte (11000000) will be stored in the first byte address.	Correct
11111111	21 The second-to-last byte is stored in this second byte address.	Correct
00001111	22 The third-to-last byte is stored in this third byte address.	Correct
00000000	23 The most-significant byte (00000000) will be stored in the last byte address.	Correct

Reset

Feedback?

PARTICIPATION ACTIVITY 2.3.6: Endianness.

1) Little-endian processors are faster than big-endian.

- ○ True
- ◉ False

Correct

Endianness does not significantly impact performance. Some processors happen to use one format, some use the other.

2) Programmers spend much time and effort focusing on endianness.

- ○ True
- ◉ False

Correct

Endianness is usually hidden from the programmer. However, knowledge of endianness is relevant when designing a processor and when learning how processors work. Endianness also matters for the (rare) cases when a programmer is doing byte-level operations on data.

3) In little-endian format, 10000000 would become 00000001.

- ○ True
- ◉ False

Correct

The bits within a byte don't change order. Only the bytes themselves are stored in different order.

Feedback?

How was this section? Provide section feedback