Exercise sheet 5 – Processor architecture

Goals:

- Synchronisation commands
- Endianness

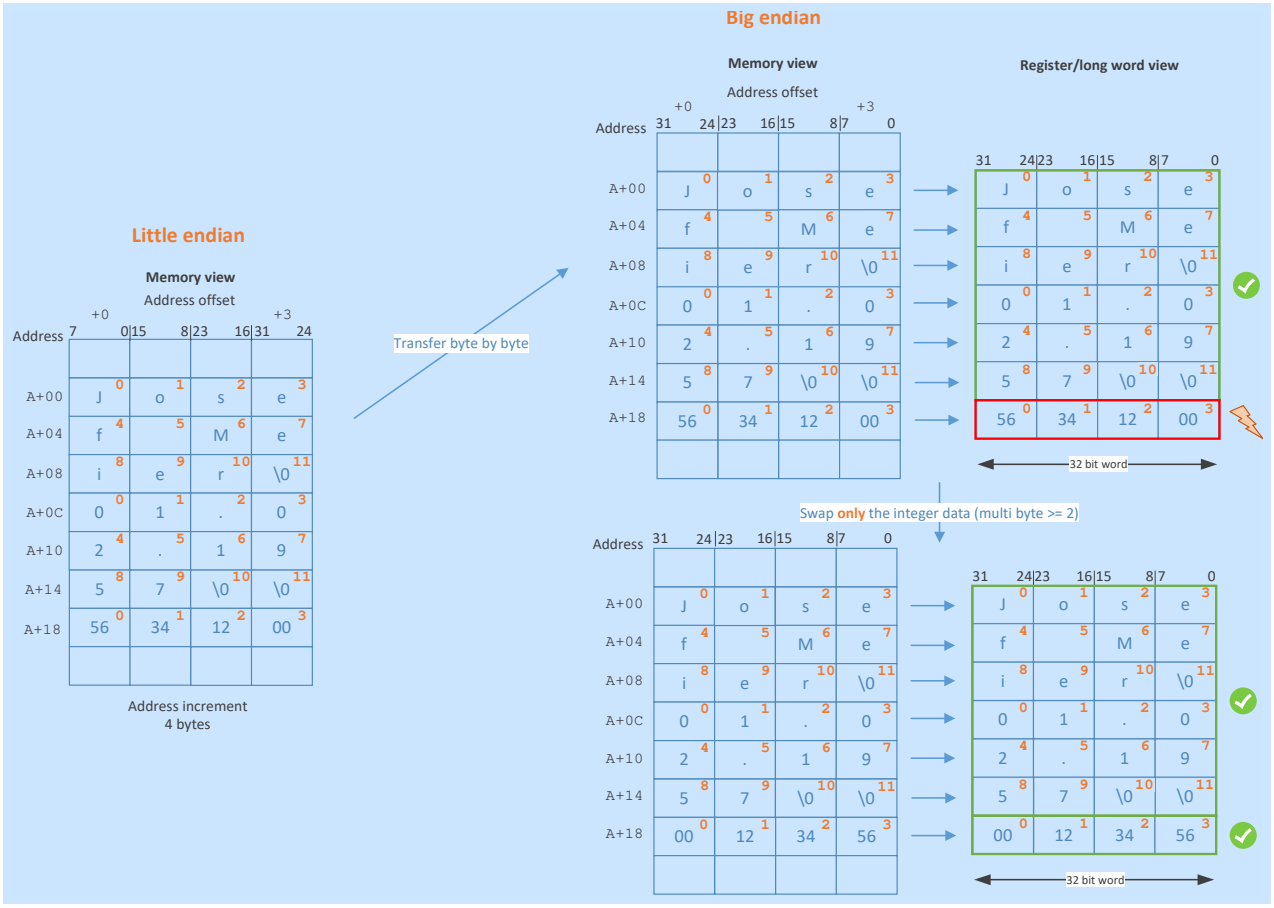
Exercise 5.1: Endianness (theoretical)

(a) The given struct `meier` is transmitted serially (byte-by-byte) from a *little-endian* to a *big-endian* architecture. Assume both are 32-bit architectures.

```
1  struct employee {
2      char name[12];     //ASCII/UTF-8 (1 byte)
3      char birthday[12]; //ASCII/UTF-8 (1 byte)
4      int32_t id;
5  };
6
7  struct employee meier = {
8      .name     = "Josef Meier",
9      .birthday = "01.02.1957",
10     .id       = 0x123456
11 };
```

Provide a solution similar to the scheme in the lecture. Which corrections may be necessary?

Proposal for solution:



Exercise 5.2: Endianness with integer (coding)

Given is a *big-endian* system program—the Java runtime environment—that transfers data via a file to a little-endian system C program. Investigate the behaviour: You may have to fix something for a correct transfer.

(a) Update the `RA_exercises` repository with `git pull`.

(b) Change into the directory `RA_exercises/sheet_05/Endianness/Java_BE`

Proposal for solution: `cd RA_exercises/sheet_05/Endianness/Java_BE`

(c) Inspect, build, and run the given Java program.

Proposal for solution:

```
make                    #build
java java_be_example    #execute
```

(d) Inspect and interpret the result file `output.txt`. *Hint: Use a HEX viewer, for example:* `xxd`.

```
xxd -c 1 output.txt #to show the byte per byte view (memory/file view) \\
xxd -c 4 output.txt #to show the word view (register view in 32 bit/4 byte mode)
```

(e) Change into the directory `RA_exercises/sheet_05/Endianness/C_LE`

Proposal for solution: `cd RA_exercises/sheet_05/Endianness/C_LE`

(f) Inspect, build, and run the given C program.

Proposal for solution:

```
make                    #build
./c_le_example          #execute
```

(g) Analyse the output of the C program. What happened? What could be the cause of this?

Proposal for solution: The C program is reading the content of *output.txt*, which was generated by the java-program. Because of the different endianness of Java (big endian) and C (little endian), the output of the C program is switched.

(h) Fix the problem in the C program, following the *TODOs*.

Proposal for solution:

```c
#include <stdio.h>  //fopen, ...
#include <stdlib.h> //EXIT_SUCCESS
#include <stdint.h> //uint8_t, uint16_t

uint16_t swap_bytes(uint16_t value_to_swap); //prototype

int main(void) {

    uint16_t value = 0;

    FILE* file = NULL;
    file = fopen("../output.txt", "rb");

    if (file == NULL) {
        printf("Error opening output.txt\n");
        return EXIT_FAILURE;
    }

    fread(&value, sizeof(uint16_t), 1, file);
    fclose(file);

    printf("Read from output.txt -> : %2x\n", value);

    //fix the byte order by calling the swap_bytes() function
    value = swap_bytes(value);

    //print the fixed value
    printf("Converted to LE -> : %2x\n", value);

    return EXIT_SUCCESS;
}


/*
We are using a union to access the uint16_t value via the byte-array (uint8_t[]).
This is working because you can only store one value at a
given time in a union
(a union is only as big as its biggest member)
and all members of that union access the same piece of memory,
but with their respective data-types.
*/
uint16_t swap_bytes(uint16_t value_to_swap) {
    union {
        uint16_t value;
```

```
45          uint8_t byte[2];
46      } SWAP_TYPE;
47
48      SWAP_TYPE.value = value_to_swap;
49
50      uint8_t tmp = SWAP_TYPE.byte[0];
51      SWAP_TYPE.byte[0] = SWAP_TYPE.byte[1];
52      SWAP_TYPE.byte[1] = tmp;
53
54      //alternatively you can do bit shift operations
55      //SWAP_TYPE.value = ((0x00FF & SWAP_TYPE.value) << 8)
56      //                | ((0xFF00 & SWAP_TYPE.value) >> 8);
57
58      return SWAP_TYPE.value;
59  }
```

(i) Build and run the C program again. Is the problem now solved?

Proposal for solution:

```
make                    #build
./c_le_example          #execute
```

Exercise 5.3: Endianness with an UTF16 character (coding)

Given is a *big-endian* UTF16 encoded character saved in a file. A C/C++ program (*little-endian*) wants to read and print the encoded character to the terminal.

(a) Change into the directory `RA_exercises/sheet_05/Endianness_UTF16/read_UTF16_character`

Proposal for solution: `cd RA_exercises/sheet_05/Endianness_UTF16/read_UTF16_character`

(b) Inspect the UTF16 character within the given file `RA_exercises/sheet_05/Endianness_UTF16/utf16_character_be.txt`.
Hint: Use a HEX viewer, for example: `xxd`.

```
xxd -c 1 utf16_character_be.txt #to show the byte per byte view (memory/file view) \\
xxd -c 4 utf16_character_be.txt #to show the word view (register view in 32 bit/4 byte
```

(c) Inspect, build, and run the given C/C++ program.

Proposal for solution:

```
make                    #build
./read_utf16_character  #execute
```

(d) Analyse the output of the C/C++ program. What happened? What could be the cause of this?

Proposal for solution: The C/C++ program is reading the content of *utf16_character_be.txt*, which contains an UTF16 character with *big-endian* encoding. That causes the output to be wrong.

(e) Fix the problem in the C/C++ program, following the *TODOs*.

Proposal for solution:

```
#include <stdlib.h> //EXIT_SUCCESS
#include <string>   //std::string, char16t
```

```cpp
#include <iostream> //std::cout

#include <locale>   // wstring_convert
#include <codecvt>  // codecvt_utf8

//prototypes
void print_utf16_string(const std::u16string& message,
                        std::u16string utf16_character);
char16_t swap(const char16_t character);

int main(void) {
    char16_t message[] = u"A nuclear power plant is ";
    char16_t utf16_character = u' ';

    //read utf16 character from file
    FILE* file = NULL;
    file = fopen("../utf16_character_be.txt", "rb");

    if (file == NULL) {
        std::cerr << "Error opening: " << "utf16_character_be.txt" << std::endl;
        return EXIT_FAILURE;
    }

    fread(&utf16_character, sizeof(uint16_t), 1, file);
    fclose(file);

    //fix the byte order by calling the swap() function
    utf16_character = swap(utf16_character);

    //print the message with the utf16 character
    print_utf16_string(message, std::u16string{utf16_character});

    return EXIT_SUCCESS;
}

void print_utf16_string(const std::u16string& message,
                        std::u16string utf16_character) {
    std::wstring_convert<std::codecvt_utf8<char16_t>, char16_t> convert;
    std::cout << convert.to_bytes(message)
              << convert.to_bytes(utf16_character) << std::endl;
}

char16_t swap(const char16_t character) {
    union {
        char16_t c;
        char bytes[2];
    } SWAP_TYPE;

    SWAP_TYPE.c = character;

    char tmp = SWAP_TYPE.bytes[0];
    SWAP_TYPE.bytes[0] = SWAP_TYPE.bytes[1];
    SWAP_TYPE.bytes[1] = tmp;

    return SWAP_TYPE.c;
}
```

(f) Build and run the C program again. Is the problem now solved?

Proposal for solution:

```
make                    #build
./read_utf16_character #execute
```