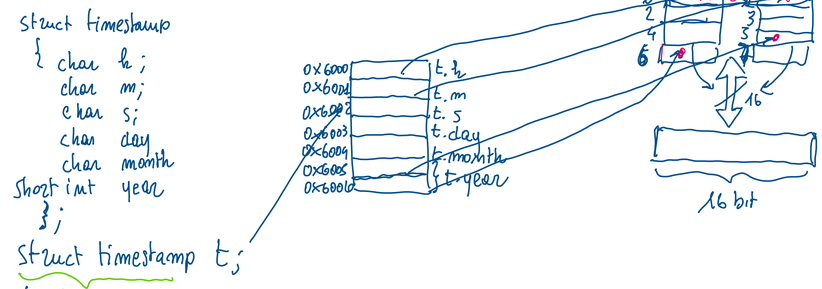
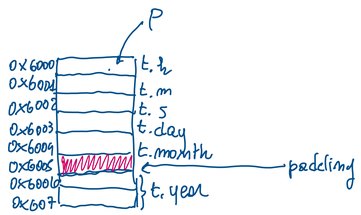
2.5 STRUCTURES - PACKING & PADDING

mercoledì 28 febbraio 2024 08:55





the buffer is "ready to send" with no copies.

```
+------------+------------+------------+------------+
|  Bank 0    |  Bank 1    |  Bank 2    |  Bank 3    |
+------------+------------+------------+------------+
| Byte 0     | Byte 1     | Byte 2     | Byte 3     | <- A single access can read/write these 4 bytes simultaneously
+------------+------------+------------+------------+
| Byte 4     | Byte 5     | Byte 6     | Byte 7     |
+------------+------------+------------+------------+
| Byte 8     | Byte 9     | Byte 10    | Byte 11    |
+------------+------------+------------+------------+
|    ...     |    ...     |    ...     |    ...     |
+------------+------------+------------+------------+


 Misaligned access of an `int` starting at address 5:

+------------+------------+------------+------------+
|  Bank 0    |  Bank 1    |  Bank 2    |  Bank 3    |
+------------+------------+------------+------------+
| Byte 4     | Byte 5*    | Byte 6     | Byte 7     |
+------------+------------+------------+------------+
               ^
             Access starts here, requiring additional operations

+------------+------------+------------+------------+
|  Bank 0    |  Bank 1    |  Bank 2    |  Bank 3    |
+------------+------------+------------+------------+
| Byte 8     | Byte  9    | Byte 10    | Byte 11    |
+------------+------------+------------+------------+
                  ^
                Access ends here, misaligned
```

Structures (structs) in C are compound data types that group together variables under a single name. These variables, known as members, can be of different types and represent a record. For example, a struct could be used to represent a date, combining an integer day, an integer month, and an integer year into a single unit.

Basic struct Declaration

Here's how you might declare a struct for a simple date:

```
struct Date
{intday;
 intmonth;
 intyear; };
```

Creating and Accessing struct Variables

To use a struct, you declare a variable of that struct type, then access its members using the dot operator:

```
Struct Date today;
today.day = 24;
today.month = 9;
today.year = 2024;
```

Memory Layout of Structs

The memory layout of a struct is sequential, with each member being stored in memory in the order declared. However, due to alignment requirements of different data types, compilers may introduce padding bytes between members or at the end of the struct to ensure that each member is aligned to its natural boundary, which can optimize memory access on many architectures.

Example Memory Map for Date Struct

Consider an architecture where integers are 4 bytes and have an alignment requirement of 4 bytes. Here's a possible memory map for the Date struct:

```
+----------------+    <- Start of struct (e.g., address 0x00)
| day (4 bytes)  |
+----------------+    <- Address 0x04
| month (4 bytes)|
+----------------+    <- Address 0x08
| year (4 bytes) |
+----------------+    <- End of struct (e.g., address 0x0C)
```

Total size: 12 bytes, with no padding required between members since each int naturally aligns on a 4-byte boundary.

Considerations with Padding

Padding can impact the size and layout of a struct. For example:

```
structMixedData
{char a; // 1 byte
intb; // 4 bytes, might be padded to align to 4-byte boundary
 charc; // 1 byte, might cause padding at the end for alignment};
```

Memory map considering padding for alignment:

```
+----------------+    <- Start of struct
| a (1 byte)     |
| Padding (3 bytes)  | <- Padding for alignment
+----------------+
| b (4 bytes)    |
+----------------+
| c (1 byte)     |
| Padding (3 bytes)  | <- Padding for alignment of the struct
+----------------+    <- End of struct
```

Total size: 12 bytes. Without the padding, the size would naively be 6 bytes, but alignment requirements increase the size to ensure that b is aligned, as well as potentially padding at the end to align the entire structure size in arrays or other uses.

Key Points

- Struct members are stored in memory in the order they are declared.
- The compiler may introduce padding between members or at the end of a struct to satisfy alignment requirements, affecting the total size.

- Accessing struct members is done via the dot operator for variables and the arrow operator (->) for pointers to structs.
- Structs are essential for grouping related data and are widely used for more complex data management in C programs, such as representing records or more complex data structures.
  Understanding structs and their memory layout is crucial for efficient C programming, especially in systems programming, embedded systems, and applications where memory layout and size are critical concerns.