Student ID: 230018374 **Date:** 03/11/2025

Practical 2 - Buddy Allocator Algorithm

Algorithm Implementation: As given to use through the `page_metadata` struct, the buddy page allocator manages memory by maintaining multiple free lists, one for each block order. When a request for pages is made, the allocator will search for a free block of at least the requested order by checking that free list. If no block of that order is found, it will attempt to find the smallest bigger order which then is recursively split into smaller blocks until a block of the desired size is obtained. The selected block is then removed from the free list and its metadata updated to reflect the new allocation. If zero-initialization is requested, the allocator clears the pages; otherwise, it searches higher-order free lists and logs an error if no suitable block is found.

Assumptions:
- `allocate_pages()` should only log a message if there's a failure when allocating. This was based on the self-test logs which were not expecting any other behaviour.
- `split_block()` and `merge_buddies()` are only going to be called internally in a way in which the programmer can assure that the passed page is free.

Optimisations:
- For both the `calculate_other_buddy_pfn()` and `split_block()` I employed bit-manipulation operations instead of multiplication for improved performance.
- When calling the `insert_free_pages()` the function will attempt to do a coalition of the inserted blocks, which makes sure to reduce fragmentation as much as possible, allowing for a faster and safer allocation as there's as little fragmentation as possible.

Challenges and Solutions:
- **Documentation and Learning**: I initially found it challenging to begin the implementation, as my notes did not provide sufficient insight into the algorithm's structure. However, by revisiting the lecture and visualising the block organisation of the allocation system, the underlying concept became progressively clearer. Although I struggled at first to grasp the relationship between the order, pages, and block sizes, I ultimately developed a solid understanding of how these components interrelate within the functionality of the algorithm.
- **Debugging**: Since I completed the entire implementation without conducting any intermediate testing, the debugging process proved to be particularly challenging. This approach required addressing all encountered issues sequentially, rather than resolving them incrementally. One of the most significant challenges I faced was a failure in the coalescing process when the order of the coalesced block was zero. This issue arose due to incorrect assertions within the `merge_buddies()` and `insert_free_pages()` functions. Through extensive use of `dprintf` statements and analysis of the `self-test` output logs, I was able to identify and resolve the underlying problem.