1. How do you access addresses in real mode?
    a. In real mode the addresses a process use are merely offsets in a certain segment and are converted to physical addresses according to the relevant segment.
    The processor identifies which register's value (code segment, data segment, stack segment, extra segment) to use as the segment base for the address according to the type of command (push or pop are stack, instructions will use code segment, data will use data segment, string operations will use extra segment), but the segment can be overridden for specific instructions by using a segment prefix byte in the instruction.
    Once the processor has the base segment address and the offset from the command it simply multiplies the base segment address by 16 and adds the offset to that, the result is the physical address (this is a 20 bit address, meaning up to 1 MB of memory is addressable).
    We can see that this design allows for many segment: offset pairs to be the same exact physical address, and since in real mode processes can access any memory a situation where two processes use the same physical memory is possible.
    The size of a segment is 64KB (16 bit address), but the fact that segmentation is used allows processes access to more than 64KB.
2. How do you access addresses in protected mode?
    a. Assuming 32 bit architecture, the 16 bit segment registers are now made up of 2 bits of requested privilege level, 1 bit table indicator (global descriptor table if bit is 1, local descriptor table otherwise), and a 13 bit index for that table.
    The processor goes to the relevant descriptor table, and uses the index to find the relevant descriptor table entry. In that entry there is a base address of a segment (32 bits), a limit value for the segment (20 bits), a privilege level for the segment (2 bits), and a few more bits.
    The processor verifies that the requested privilege level (and the privilege level of the code itself, in code segment register) are less than or equal to the segment privilege level in the descriptor table entry.
    One of the extra bits in the entry is a granularity bit that can set the granularity of the limit to be 1 or 4 KB (depending on the bit's value), meaning the limit can be exactly its value (granularity 1) or its value times 4KB (granularity 4KB). Using this we verify that the offset (the address used in the instruction, which is 32 bit of course) is smaller than the limit of the segment in the entry.
    Now we take the base address of the segment from the entry and add to it the offset, this address is the counterpart of the physical address in real mode, but in protected mode is merely the input to the paging unit (if it is enabled). Meaning that address is a virtual address.
    This address is split into three parts: page directory entry (10 bits), page table entry (10 bits), offset (12 bits). The CPU holds a register that has the base address of the page directory entry and we use the first 10 bits as an index to the page directory, the entry of the directory we get to holds the base address of a page table. We use the second 10 bits as an index to the page table and get an entry that holds the base address of a page. We use our remaining bits (the offset) and add them to the base page address to get the physical address.

During the process of paging the system can restrict a process from accessing pages that aren't his or he doesn't have privilege for. Pages can also be stored on ROM as well as RAM and if a page from ROM it will be "paged" into the RAM.

b. 64 bit architecture mostly doesn't use segmentation (when in long mode) and instead only uses paging. The paging hierarchy (how many page tables deep we go) is extended to 4 (instead of 2).

3. Why does real mode still exist?
    a. During the very beginning of the startup of a computer it uses real mode, because the page directory and the relevant registers haven't yet been initialized so it can't run in protected mode yet. One of the first things modern CPUs do is initialize everything needed and go into protected mode.
    b. Another reason is backwards compatibility, some software (for example BIOS) assumes it runs in real mode, so newer processors also supported that older software.
4. Can we make an architecture that doesn't support real mode?
    a. Theoretically yes, we can define to the processor default values for page translation, but in essence we still get real mode, because the processor must start executing at a specific known address. We can have the processor do page translation for that address or just use it directly, it really doesn't matter.