

Virtualization

Pre-lab Questions — #1

Point total: 7

All homework assignments are weighted equally in the final grade. Point values are unique to each lab assignment.

Note: For all problems which ask you to explain your reasoning or show your work, you do not need to show every step of each calculation, but the answer should include an explanation *written with words* of what you did. Even when work is not required to be shown, it's a good idea to include anyways so that you can earn partial credit.

1. Question [1 point] What does the `cpuid` assembly instruction do in this function?

Solution:

The `cpuid` assembly instruction provides details about the processor. When `cpuid` is called with an initial input value of 0 in the EAX register, it returns the highest function number the CPU supports and the vendor ID string. This 12-character ASCII string identifies the manufacturer and is distributed across the EBX, EDI, and ECX registers.

2. Question [1 point] How is the function providing values back to you to use?

Solution:

The `cpuid` function retrieves information directly into the CPU registers EAX, EBX, ECX, and EDI using inline assembly. After the instruction is executed, it utilizes the provided pointers (`eaxp`, `ebx`, `ecx`, `edi`) to relay these values to specific memory locations. This method allows the function, `vmx_check_support`, to access the results of the `cpuid` instruction by referencing the variables 'eax', 'ebx', 'ecx', and 'edi'.

3. Question [1 point] What are the results of `eax`, `ecx`, and `ebx` values in hexadecimal?

Solution:

The hexadecimal results for the registers are:

- EAX (`rax`): 0x8003ffffe60
- ECX (`rcx`): 0x444d4163
- EBX (`rbx`): 0x0

4. Question [1 point] Now examine the values of these variables as strings. Hint: look at the values in hexadecimal and translate them to strings, in the order `ebx`, `edx`, `ecx`. What do you observe? The Wikipedia page for the `cuid` instruction may help you interpret this output.

Solution:

1. Given the register values after the `cuid` call for an AMD processor:

- `EBX (rbx)`: `0x68747541` → Translates to ASCII characters "Auth"
- `EDX (rdx)`: `0x69746e65` → Translates to ASCII characters "entic"
- `ECX (rcx)`: `0x444d4163` → Translates to ASCII characters "cAMD"

Collating them in the order of `EBX`, `EDX`, and `ECX`, we derive the vendor string: "AuthenticAMD". This string is an identifier for AMD processors.

5. Question [1 point] There is a reference in each `Env` struct for another struct called `VmxGuestInfo`. What kind of information does this struct hold?

Solution:

`phys_sz` 64-bit value, indicating storage size.

`vmcs` Points to the VMCS, which manages VM state and controls its execution.

`exception_bmap` Bitmap indicating which exceptions the hypervisor handles.

`io_bmap_a`, `io_bmap_b` I/O bitmaps that control guest access to I/O ports. Different bitmaps may represent different port sets.

`msr_count` Count of MSRs being managed.

`msr_host_area`, `msr_guest_area` Memory areas for host and guest MSRs, specialized CPU registers.

`vcpunum` Identifier for the associated virtual CPU.

The `VmxGuestInfo` struct holds data for virtual machine management in VMX, including exception tracking, I/O access, and MSR handling.

6. Question [1 point] From this Intel guide, find out what the `vmcs` pointer in this struct stands for, and what its purpose it serves.

Solution: The VMCS is central to Intel's VMX virtualization. It holds state areas for the host and the guest, ensuring smooth transitions between the Virtual Machine Monitor (VMM) and the virtual machine. It also dictates how the virtual machine behaves during execution. The `vmcs` pointer in `VmxGuestInfo` tracks the VMCS location for a specific guest, ensuring proper virtual machine management and control.

7. Question [1 point] What assembly instruction initializes the `vmcs` pointer? In other words, how do we change the `vmcs` pointer?

Solution:

To change or load a new address into the VMCS pointer, the `VMPTRLD` instruction is utilized. This instruction effectively updates the VMCS pointer to reference a new VMCS region, thereby allowing dynamic management of different VMCS regions for varied virtual machine contexts.