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 calling parameter the CPU supports and the vendor ID string. This 12-character ASCII string identifies the manufacturer and is distributed across the EBX, EDX, 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 EDX using inline assembly. After the instruction is executed, it utilizes the provided pointers (`eaxp`, `ebxp`, `ecxp`, `edxp`) 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 'edx'.

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): 0x8003fffe60`
- `ECX (rcx): 0x444d4163`
- `EBX (rbx): 0x0`

According to the CPUID wikipedia page, the return of the EAX register should hold the value of the highest basic calling parameter, but the AMD processor makes it hard to confirm.

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 cpuid instruction may help you interpret this output.

Solution: Given the hexadecimal values stored in the registers,the converted ASCII representations is as follows:

1. **RCX (rcx):** 0x444d4163
   - 0x44 translates to "D"
   - 0x4D translates to "M"
   - 0x41 translates to "A"
   - 0x63 translates to "c"

   Together, these bytes form the string "DMAc".
2. **RDX (rdx):** 0x69746e65
   - 0x69 translates to "i"
   - 0x74 translates to "t"
   - 0x6e translates to "n"
   - 0x65 translates to "e"

   Together, these bytes form the string "itne".
3. **RSI (rsi):** 0x68747541
   - 0x68 translates to "h"
   - 0x74 translates to "t"
   - 0x75 translates to "u"
   - 0x41 translates to "A"

   Together, these bytes form the string "htuA".

When we combine the strings from the RCX, RDX, and RSI registers in that order, we get DMAcitnehtuA. Reversed, this becomes AuthenticAMD, a signature for AMD processors. Tthe value we expected to be in the RBX register was found in the RSI register, while RBX held a value of 0x0, providing no useful information. We think this discrepancy might stem from using an AMD rather than an Intel processor.

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, the storage size.

`vmcs` Points to the VMCS, which manages the 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.

`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 it 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. 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.