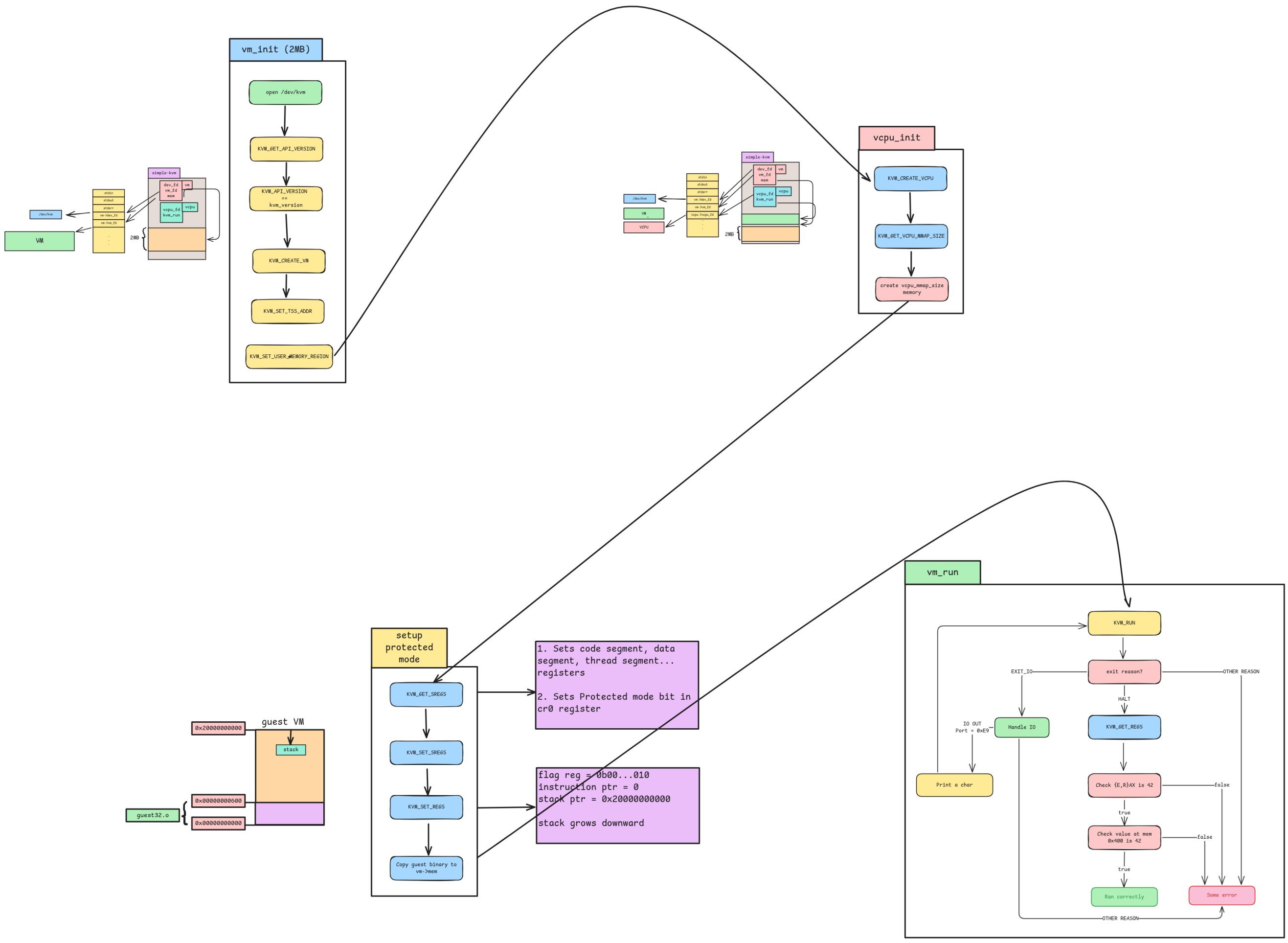
Part 1a

1. Flow chart for protected mode execution



2. Explain the code snippets

a.

```
extern const unsigned char guest64[], guest64_end[];
```

- Here we are **declaring** 2 char pointers which are **defined** somewhere in external file
- By deducing from makefile and project structure, we can find they point to start and end of **BLOB (Binary large object)** for the guest code(which is `.img` file)
- another code snippet used these pointers `memcpy(vm→mem, guest64, guest64_end - guest64);` to copy whole binary into the memory region allocated for the Virtual Machine.

---

b.

```
pml4[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pdpt_addr;

pdpt[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pd_addr;

pd[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | PDE64_PS;

sregs→cr3 = pml4_addr;

sregs→cr4 = CR4_PAE;

sregs→cr0 = CR0_PE | CR0_MP | CR0_ET | CR0_NE | CR0_WP | CR0_AM | CR0_PG;

sregs→efer = EFER_LME | EFER_LMA;
```

- This code snippet used to set up 4-level 64 bit paging
- Macros `PDE64_PRESENT` ensures that page table entry is valid, `PDE64_RW` ensures that the page is both readable and writable, `PDE64_USER` ensures that page is user accessible, and `PDE64_PS` ensures page size is 1 GB
- `pml4[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pdpt_addr;`
  - `pml4` is top-level page table
  - `0` index points to page directory pointer table (`pdpt_addr`)
- `pdpt[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pd_addr;`
  - `pdpt` is second level page table (Page directory pointer table)
  - `0` index points to page directory(`pd_addr`)
- `pd[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | PDE64_PS;`
  - `pd` is third level page table
  - `0` index points to a page
- `sregs` → refer to special registers
  - `cr3` (page table base) → register will contain address of Top level Page table (`pml4_addr`)
  - `cr4` (page table extension) → register when assigned `CR4_PAE` will enable Physical Address Extension (for 64-bit paging). Now Physical address will be of size 52-bit.
  - `cr0` (core control) →
    - `CR0_PE` → this will enable Protected Mode
    - `CR0_MP` →
    - `CR0_ET` →
    - `CR0_NE` → this will enable handling of native F
    - `CR0_WP` → this will kernel pages **write protect**
    - `CR0_AM` →
    - `CR0_PG` → this will enable paging

---

c.

```
vm->mem = mmap(NULL, mem_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
    MAP_ANONYMOUS | MAP_NORESERVE, -1, 0);
madvise(vm->mem, mem_size, MADV_MERGEABLE);
```

- This code is used to allocate VM memory inside `simple-kvm`
- `vm->mem = mmap(NULL, mem_size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS | MAP_NORESERVE, -1, 0);`
  - this will create a `mem_size` memory region which is **read and write protected**, and is **private**, not file-backed(**anonymous**) and will not commit swap space before told to do so(**nopreserve*)
- `madvise(vm->mem, mem_size, MADV_MERGEABLE);`
  - `madvise` is a syscall to advice to OS to make this region **mergeable** by enabling **KSM**(Kernel Samepage Merging) because of which pages identical to multiple **VMs** will be merged for efficient memory usage

---

d.

```
case KVM_EXIT_IO:
if (vcpu->kvm_run->io.direction == KVM_EXIT_IO_OUT &&
    vcpu->kvm_run->io.port == 0xE9) {
char *p = (char *)vcpu->kvm_run;
fwrite(p + vcpu->kvm_run->io.data_offset,
       vcpu->kvm_run->io.size, 1, stdout);
fflush(stdout);
continue;
}
```

- When a **VM** exits the `VMX` mode via I/O (using `in` and `out` assembly instructions), `simple-kvm` will be informed about that (`KVM_EXIT_IO`).
- If `out` instruction was executed then `io.direction` will be `KVM_EXIT_IO_OUT` else for `in` it will be `KVM_EXIT_IO_IN`
- the port is specified in `out` or `in` instruction which can be retrieved by host using `io.port`
- Since `out` is called so the `simple-kvm` writes to address given by adding `io.data_offset` to `kvm_run` address. This will print messages from guest to host console.

---

e.

```
memcpy(&memval, &vm->mem[0x400], sz);
```

- This will copy data stored at guest's memory at address `0x400` to `memval` , `sz` represents size of data.
- Then the `simple-kvm` will test if the `guest` halted correctly by checking if value at this memory is `42` .