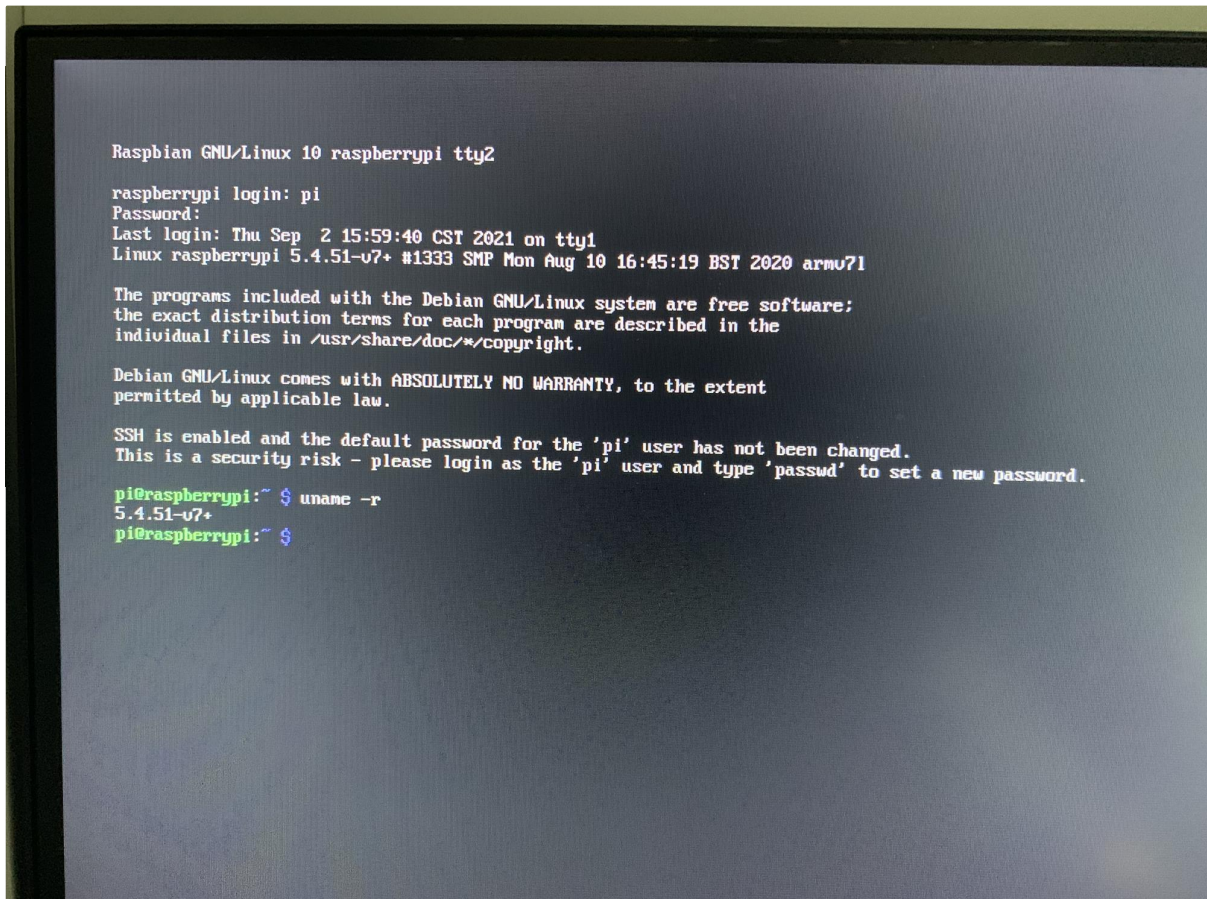


# Lab 8

## Q1

Can you prove that (1) you have replaced the kernel (with "uname -r" or other approaches)

The origin output of `uname -r` is

A photograph of a terminal window on a Raspberry Pi. The terminal shows the login process for the 'pi' user. After the login, the user runs the command 'uname -r'. The output of this command is '5.4.51-v7+', which is displayed on the line immediately following the command prompt. The terminal text is as follows:

```
Raspbian GNU/Linux 10 raspberrypi tty2
raspberrypi login: pi
Password:
Last login: Thu Sep  2 15:59:40 CST 2021 on tty1
Linux raspberrypi 5.4.51-v7+ #1333 SMP Mon Aug 10 16:45:19 BST 2020 armv7l

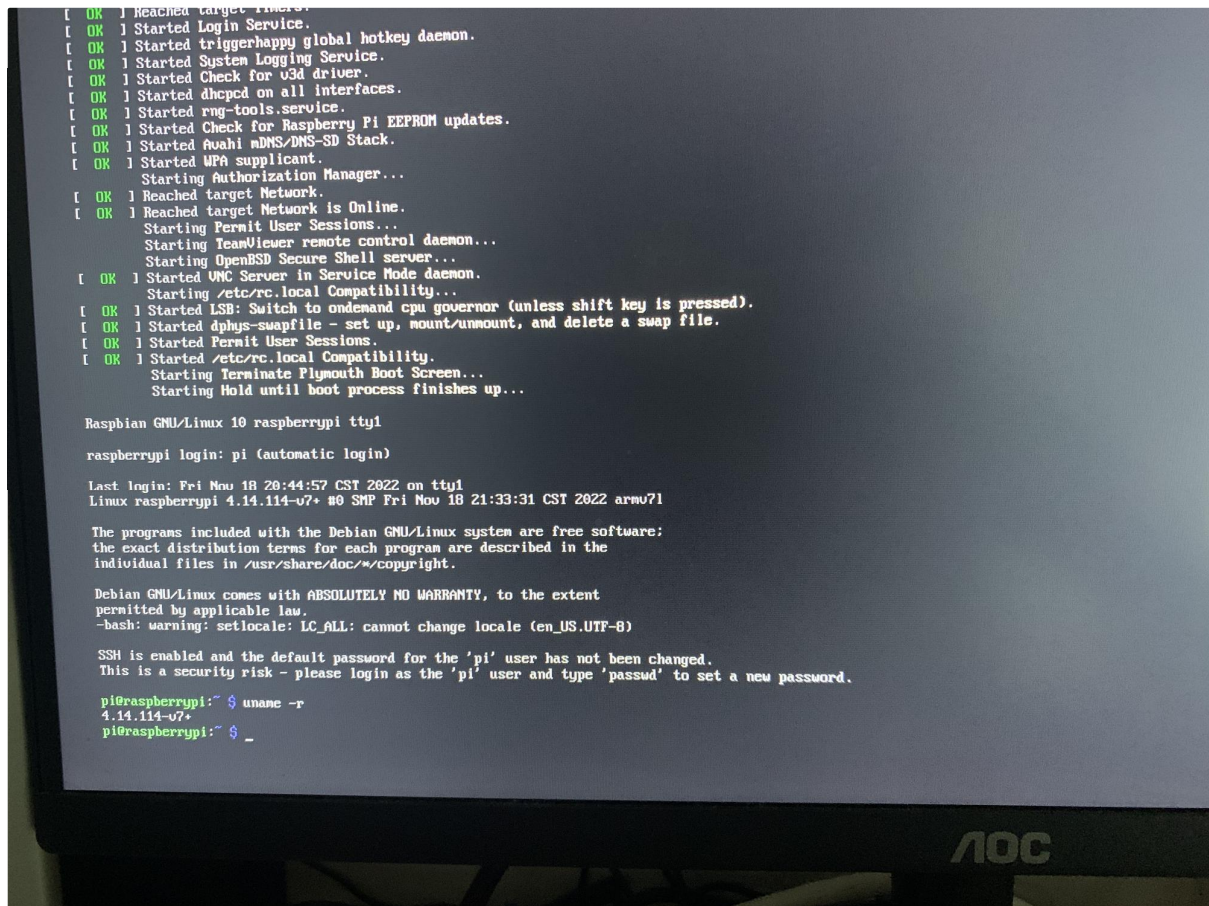
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

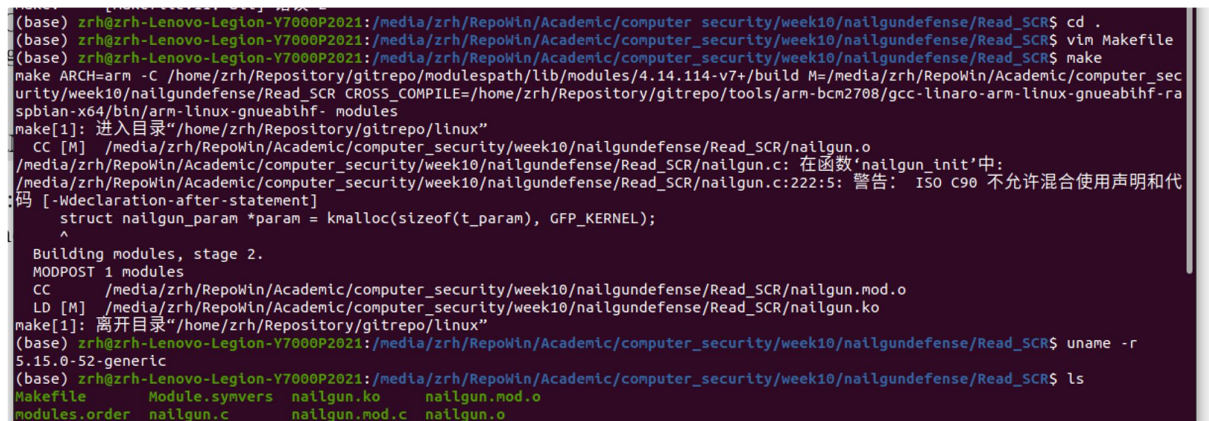
pi@raspberrypi:~$ uname -r
5.4.51-v7+
pi@raspberrypi:~$
```

After compiling and replacing the kernel, the output is



(2) you have built the nailgun module with new headers

After building the nailgun module, we got the following output and the `nailgun.ko` file




Q2

Can you run the Nailgun Attack on your new kernel

Yes, and the `dmesg` & `uname -r` output is





```

[ 3.350000] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[ 3.365455] UFS: Mounted root (ext4 filesystem) readonly on device 179:2.
[ 3.383240] devtmpfs: mounted
[ 3.396353] Freeing unused kernel memory: 1024K
[ 3.460767] usb 1-1.3: New USB device found, idVendor=c0f4, idProduct=04c0
[ 3.475262] usb 1-1.3: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 3.490175] usb 1-1.3: Product: usb keyboard
[ 3.501945] usb 1-1.3: Manufacturer: SZH
[ 3.524505] input: SZH usb keyboard as /devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3/1-1.3.1.0/0003:C0F4:04C0.0001/input/input0
[ 3.611978] hid-generic 0003:C0F4:04C0.0001: input,hidraw0: USB HID v1.10 Keyboard [SZH usb keyboard] on usb-3f980000.usb-1.3/input0
[ 3.641011] input: SZH usb keyboard as /devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3/1-1.3.1.1/0003:C0F4:04C0.0002/input/input1
[ 3.731673] hid-generic 0003:C0F4:04C0.0002: input,hidraw1: USB HID v1.10 Device [SZH usb keyboard] on usb-3f980000.usb-1.3/input1
[ 3.963160] systemd[1]: System time before build time, advancing clock.
[ 4.001375] usb 1-1.1.1: new high-speed USB device number 5 using dwc_otg
[ 4.032144] systemd[1]: systemd 241 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSUINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +AC
crachy=hybrid)
[ 4.072779] systemd[1]: Detected architecture arm.
[ 4.132166] systemd[1]: Set hostname to <raspberrypi>.
[ 4.171857] usb 1-1.1.1: New USB device found, idVendor=0424, idProduct=7800
[ 4.180454] usb 1-1.1.1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
[ 4.472391] libphy: lan78xx-mlbous: probed
[ 4.510258] lan78xx 1-1.1.1.0 (unnamed net device) (uninitialized): int usb period 64
[ 4.532091] systemd[1]: File /lib/systemd/system/systemd-journald.service:12 configures an IP firewall (IPAddressDeny=any), but the local system does not s
[ 4.560046] systemd[1]: Proceeding WITHOUT firewalling in effect! (This warning is only shown for the first loaded unit using IP firewalling.)
[ 4.838067] systemd[1]: /etc/systemd/system/teamviewerd.service:8: PIDFile= references path below legacy directory /var/run/, updating /var/run/teamviewerd
cordingly.
[ 5.140238] random: systemd: uninitialized urandom read (16 bytes read)
[ 5.180212] random: systemd: uninitialized urandom read (16 bytes read)
[ 5.198048] systemd[1]: Listening on udev Control Socket.
[ 5.225742] random: systemd: uninitialized urandom read (16 bytes read)
[ 5.243442] systemd[1]: Listening on Journal Socket (/dev/log).
[ 5.294186] systemd[1]: Condition check resulted in Journal Audit Socket being skipped.
[ 5.304136] systemd[1]: Set up automount Arbitrary Executable File Formats File System Automount Point.
[ 6.213902] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 6.372883] systemd-journald[106]: Received request to flush runtime journal from PID 1
[ 9.283450] random: crng init done
[ 9.283459] random: 7 urandom warning(s) missed due to ratelimiting
[ 9.612706] uart-pi011 3f201000.serial: no DMA platform data
[ 10.201186] Adding 102396k swap on /var/swap. Priority: -2 extents:1 across:102396k SSFS
[ 341.810509] nailgun: loading out-of-tree module taints kernel.
[ 341.811013] Nailgun Attack Start
[ 341.811066] Using smp_call_function
[ 341.811080] Step 1: Unlock debug and cross trigger registers
[ 341.811088] Step 2: Enable halting debug
[ 341.811092] Step 3: Halt the target processor
[ 341.811098] Step 4: Wait the target processor to halt
[ 341.811102] Step 5: Save context
[ 341.811108] Step 6: Switch to EL3
[ 341.811113] Step 7: Read SCR
[ 341.811118] Step 8: Restore context
[ 341.811124] Step 9: Send restart request to the target processor
[ 341.811129] Step 10: Wait the target processor to restart
[ 341.811136] All done! The value of SCR is 0x00000131
pi@raspberrypi:~/Desktop/lab8 $ uname -r
4.14.114-v7+
pi@raspberrypi:~/Desktop/lab8 $ _

```

### Q3

With the provided source codes, can you explain the process of traslating an IPA, 0x40030000+“last 3 numbers of your student ID”, to the same value of PA?

My **SID** is **11913008** , so the **IPA** is **0x40033008**

0b0100\_0000\_0000\_0011\_0000\_0000\_0000\_1000

I assume that my translation table just like the one in the lecture



## Design: Example

Here is one example of the table layout in 0x0 ~ 0xFFFF\_FFFF (only invalid dbg)

VTTBR: point to area0

area0:

0x0000\_0000 ~ 0x3FFF\_FFFF: 1GB block  
0x4000\_0000 ~ 0x7FFF\_FFFF: table, point to area1  
0x8000\_0000 ~ 0xBFFF\_FFFF: 1GB block  
0xC000\_0000 ~ 0xFFFF\_FFFF: 1GB block

area1:

0x4000\_0000 ~ 0x401F\_FFFF: table, point to area2  
0x4020\_0000 ~ 0x403F\_FFFF: 2MB block  
0x4040\_0000 ~ 0x405F\_FFFF: 2MB block

...

0x7E00\_0000 ~ 0x7FFF\_FFFF: 2MB block

area2:

0x4000\_0000 ~ 0x4000\_0FFF: 4KB Page

...

0x4002\_F000 ~ 0x4002\_FFFF: 4KB Page

0x4003\_0000 ~ 0x4003\_0FFF: Invalid (0x0)

0x4003\_1000 ~ 0x4003\_1FFF: 4KB Page

0x4003\_2000 ~ 0x4003\_2FFF: 4KB Page

...

0x401F\_0000 ~ 0x401F\_FFFF: 4KB Page

---

Now we know the following values

- IPA : 0x40033008
- BADDR in VTTBR : 0x32000000
- And the whole translation table layout

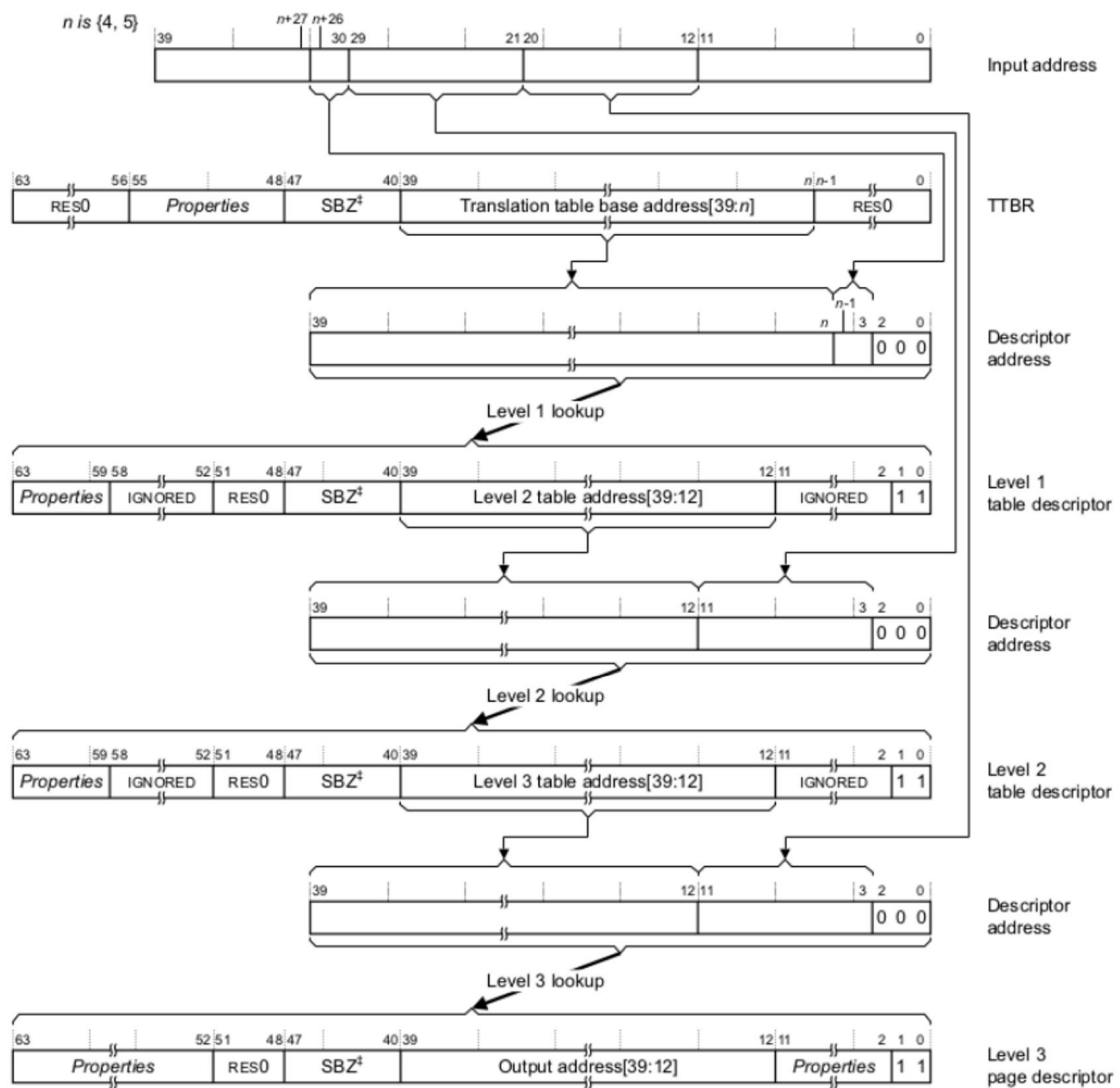


Figure 13: A translation example

According to the walks in lab, the PA can be found by following steps.

[n = 5]

So we can get the following pagetable structure.

## The number of each area

The page size is 4KB

area 0 : [n + 26] ~ 30 ⇒ 31 ~ 30, 2 bits to represent the area 0

⇒  $2^2 = 4$  entries(descriptors)

only use 1 page table

⇒  $1 * PageSize = 2^{12} = 0x1000$

area 1 : [n + 26] ~ 21 ⇒ 31 ~ 21, 11 bits to represent the area 1

$\Rightarrow 2^{11}$  entries(descriptors)

Use 4 page tables(because of 4 entries only in area 0)

$$\Rightarrow 4 * PageSize = 2^{14} = 0x4000$$

area 2 :  $[n + 26] \sim 12 \Rightarrow 31 \sim 12$ , 20 bits to represent the area 2

$\Rightarrow 2^{20}$  entries(descriptors)

Use  $2^{11}$  page

tables(because of  $2^{11}$  entries only in area 1)

$$\Rightarrow 2^{11} * PageSize = 2^{23} = 0x800000$$

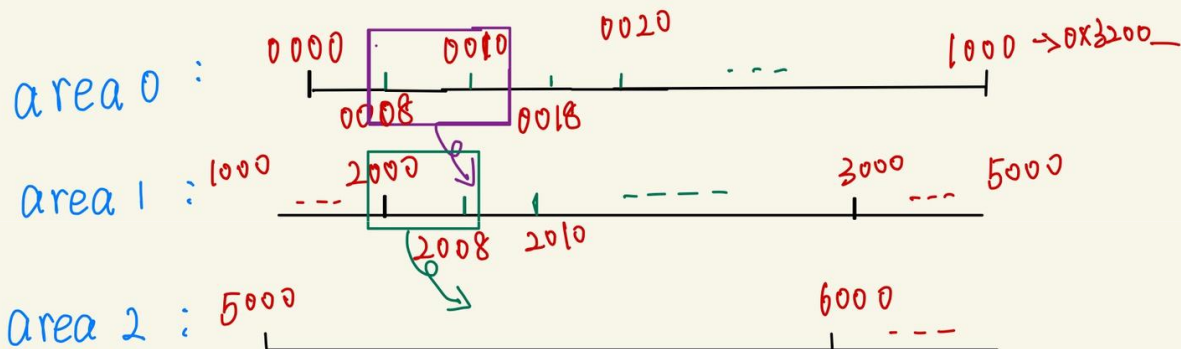
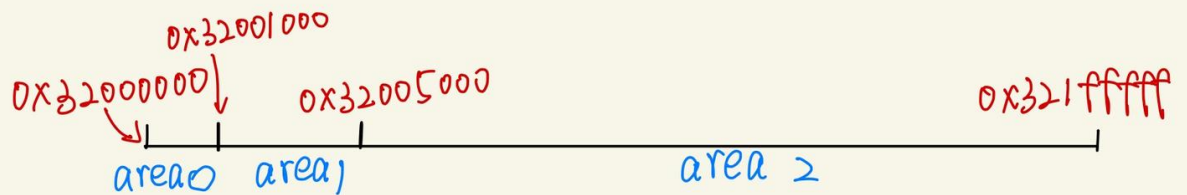
Since  $0x800000 > 2MB$  we cannot store all the descriptors. So the area 3 isn't store all the descriptors, the base address of different areas are listed followed

- area 0 (Translation table base address) :  $0x32000000$
- area 1 (Level 2 table address) :  $0x32001000$
- area 2 (Level 3 table address) :  $0x32005000$

Each descriptor is 8 bytes

$0x4000-0000 \sim 0x401F-FFFF$

   $0x4000-0000 \sim 0x7FFF-FFFF$  └─┬─┘ descriptor



The whole structure is listed before, the whole walks are listed followed

1. Access VTTBR, the value is Translation table base address, that is Descriptor address → 0x32000000
2. Access area 0, undergoing Level 1 lookup

3. Get the goal **descriptor** in Level 1, the descriptor **address (second descriptor)** is `0x3200_0008` , the **value** is point to the Level 2 table `0x3200_2000` , last two bits is `11`
4. Access **area 1**, undergoing Level 2 lookup
5. Get the goal **descriptor** in Level 2, the descriptor **address(first descriptor)** is `0x3200_2000` , the **value** is point to the Level 3 table `0x3200_5000` , last two bits is `11`
6. Access **area 2**, undergoing Level 3 lookup
7. Get the goal **descriptor** in Level 3, the descriptor **address(33rd descriptor)** is `0x3200_5000 + 0x33 * 8` bytes = `0x3200_5000 + 0x198 = 0x3200_5198` , the **value** is output address `0x40033000` (IPA has same value of PA) , last two bits is `11`
8. Access `0x40033000` , and shift 8 bytes to get the same value of IPA, that is `0x40033008`

## Q4

With the provided source codes, can you explain the process of traslating an IPA, 0x40000000+“last 7 numbers of your student ID”, to the same value of PA? (e.g., if your ID is 12150073, then you should translate 0x42150073).

My **SID** is `11913008` , so the **IPA** is `0x41913008`

So the process can be listed below

1. Access **VTTBR** , the value is **Translation table base address, that is Descriptor address**→ `0x32000000`
2. Access **area 0**, undergoing Level 1 lookup
3. Get the goal **descriptor** in Level 1, the descriptor **address (second descriptor)** is `0x3200_0008` , the **value** is pointed to the Level 2 table `0x3200_2000` , last two bits is `11`
4. Access **area 1**, undergoing Level 2 lookup
5. Get the goal **descriptor** in Level 2, the descriptor **address(9th descriptor)** is `0x3200_2000 + 9 * 8` bytes = `0x3200_2000 + 0x48 = 0x3200_2048` , the **value** is pointed to the output address `0x4180_0000` , last two bits is `01` , which means its a block
6. Access `0x4180_0000` , and shift `0x11_3008` bytes to get the same value of IPA, that is `0x41913008`



