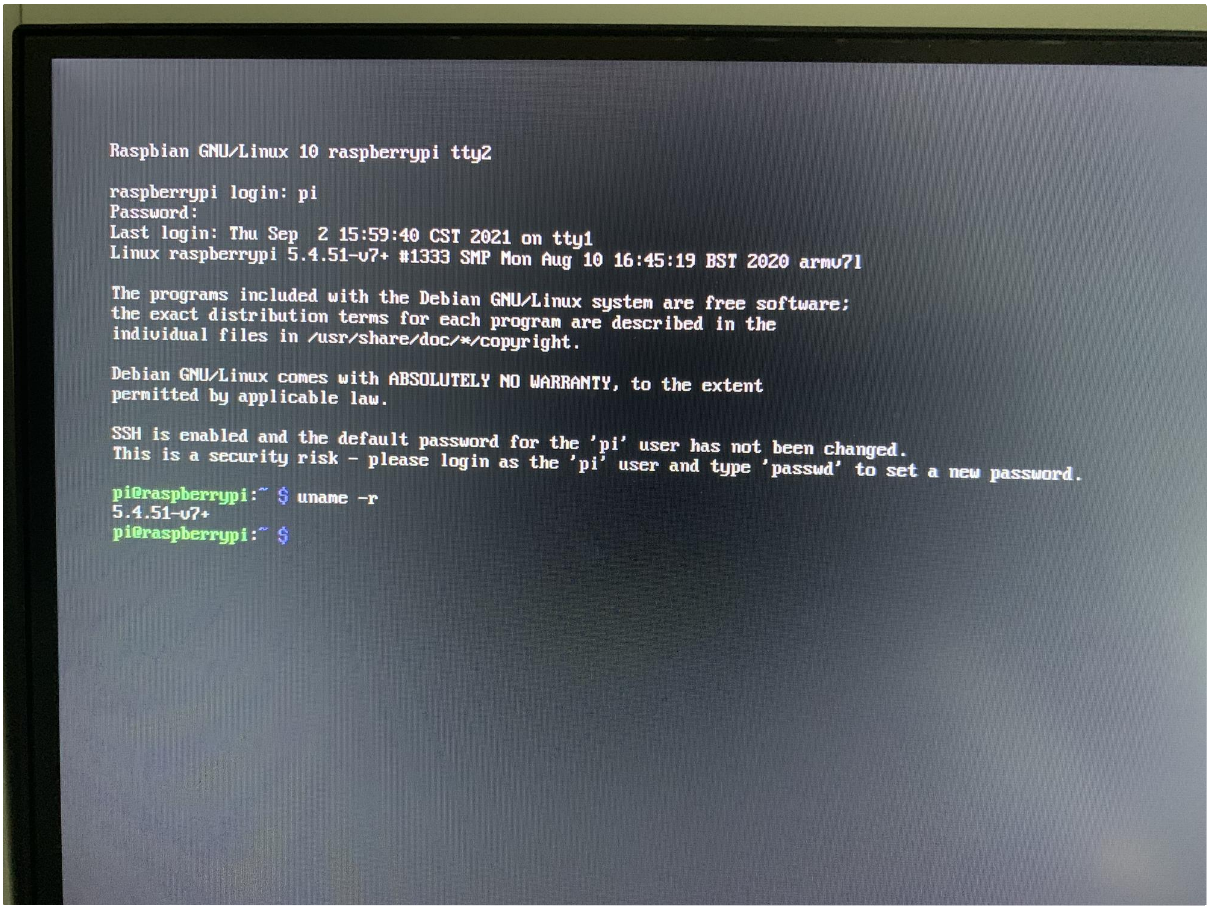
Lab 8

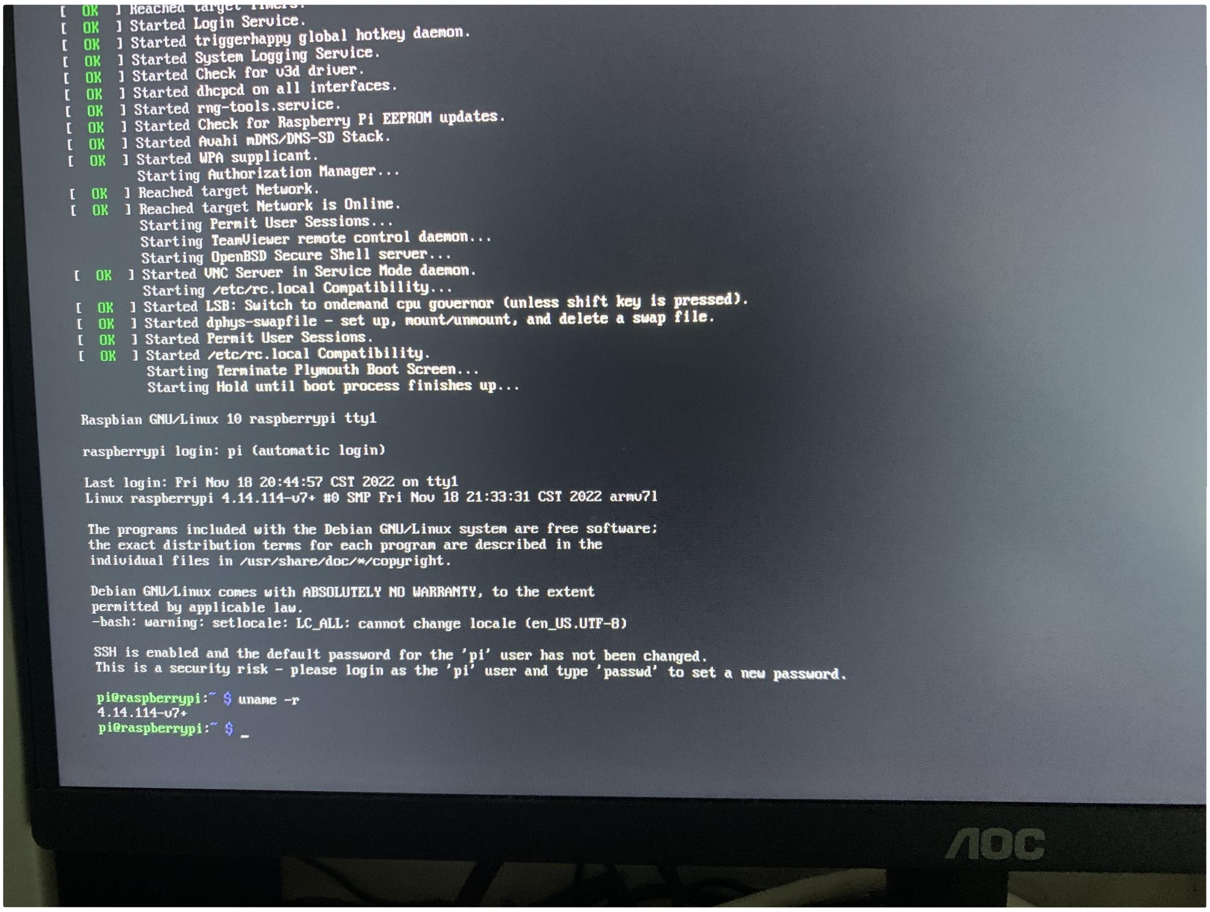
Q1

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

The origin output of `uname -r` is

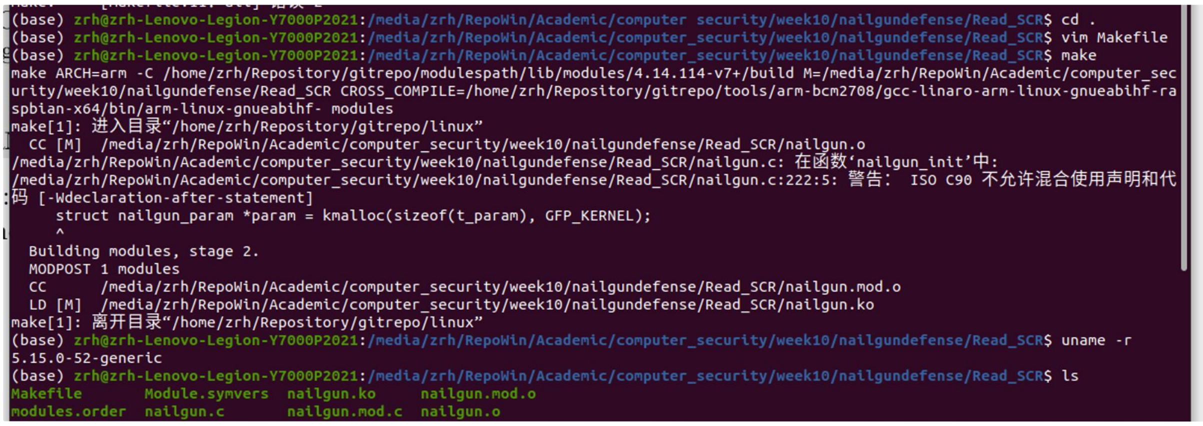


After compiling and replacing the kernel, the output is



> (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



```
(base) zrh@zrh-Lenovo-Legion-Y7000P2021:/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR$ cd .
(base) zrh@zrh-Lenovo-Legion-Y7000P2021:/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR$ vim Makefile
(base) zrh@zrh-Lenovo-Legion-Y7000P2021:/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR$ make
make ARCH=arm -C /home/zrh/Repository/gitrepo/modulespath/lib/modules/4.14.114-v7+/build M=/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR CROSS_COMPILE=/home/zrh/Repository/gitrepo/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf- modules
make[1]: 进入目录“/home/zrh/Repository/gitrepo/linux”
  CC [M]  /media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR/nailgun.o
/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR/nailgun.c: 在函数‘nailgun_init’中:
/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR/nailgun.c:222:5: 警告： ISO C90 不允许混合使用声明和代码 [-Wdeclaration-after-statement]
     struct nailgun_param *param = kmalloc(sizeof(t_param), GFP_KERNEL);
     ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR/nailgun.mod.o
  LD [M]  /media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR/nailgun.ko
make[1]: 离开目录“/home/zrh/Repository/gitrepo/linux”
(base) zrh@zrh-Lenovo-Legion-Y7000P2021:/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR$ uname -r
5.15.0-52-generic
(base) zrh@zrh-Lenovo-Legion-Y7000P2021:/media/zrh/RepoWin/Academic/computer_security/week10/nailgundefense/Read_SCR$ ls
Makefile       Module.symvers   nailgun.ko     nailgun.mod.o
modules.order  nailgun.c        nailgun.mod.c  nailgun.o
```

Q2

> Can you run the Nailgun Attack on your new kernel

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

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

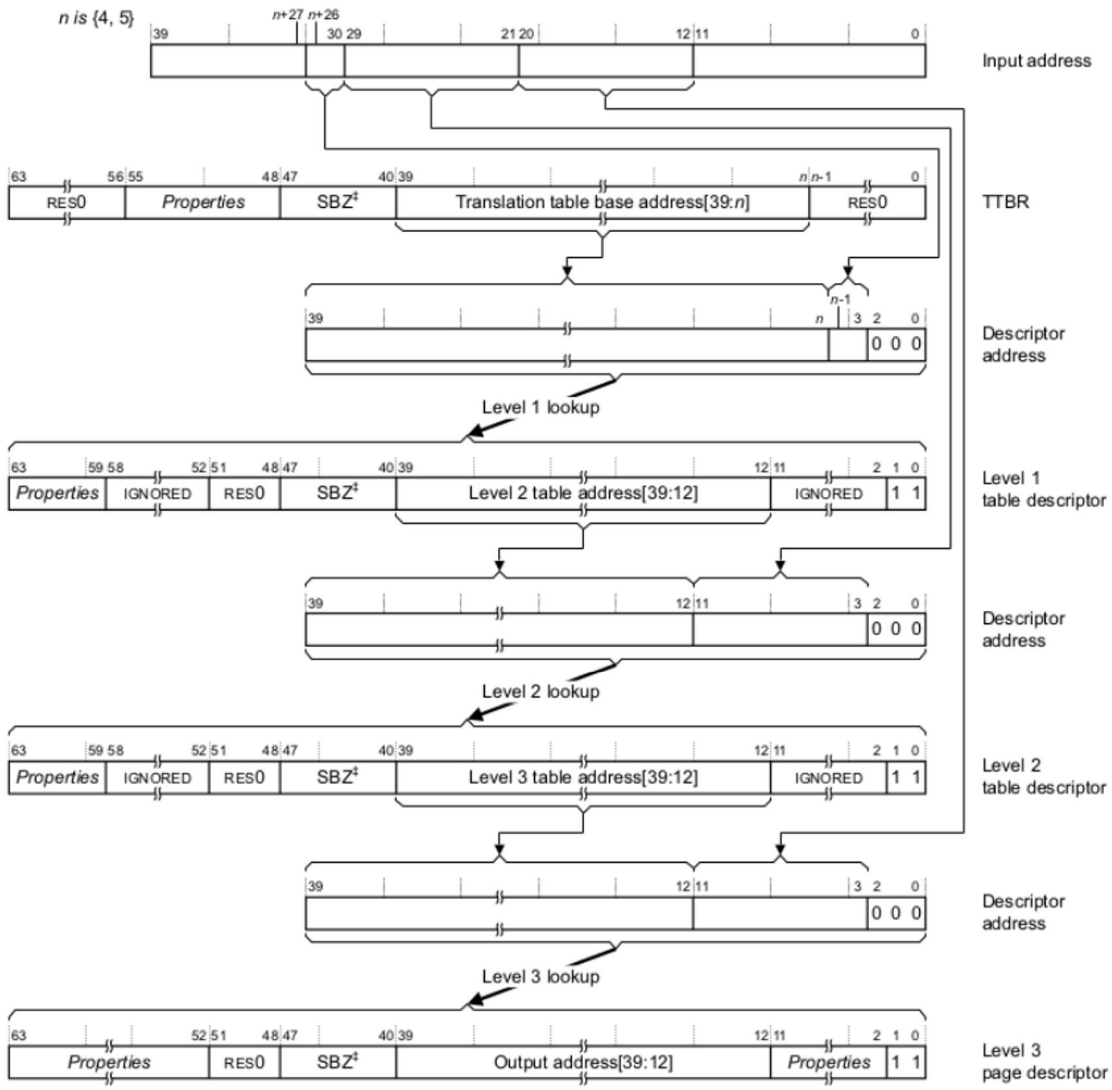


Figure 13: A translation example

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] ~ 12 ⇒ 31 ~ 12, 20 bits to represent the area 2

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

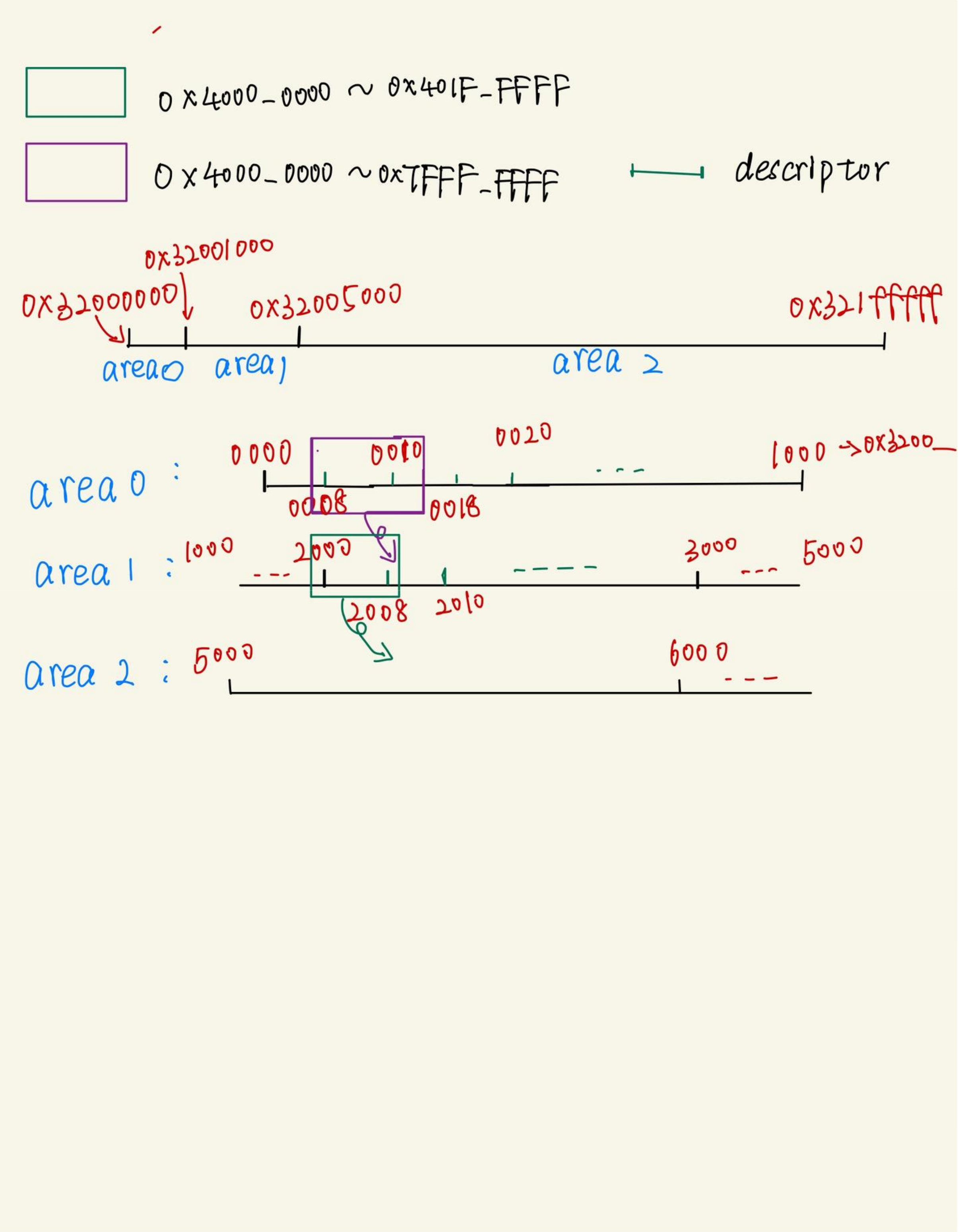
Use 2^{11} page
tables(because of 2^{11} entries only in area 1)

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



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`