운영체제입문 HW2

성균관대학교 인공지능융합전공 2019311492 이나은

#1. Segmentation

1) Different address space size and physical memory size

위의 configuration을 충족하기 위해 -a, -p의 태그 값만 달리하여 실험을 수행하였다. 우선 첫 번째 케이스에서는 address space size가 16, physical memory size가 32이다.

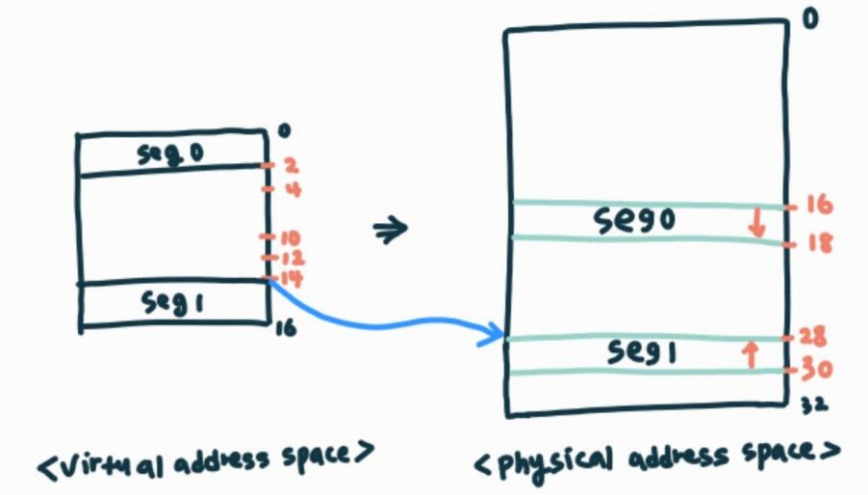
```
C:\Users\이나은\Downloads\OS_HW_2>python segmentation.py -s 20 -a 16 -p 32 -b 16 -l 2 -B 30 -L 2 -c
ARG seed 20
ARG address space size 16
ARG phys mem size 32

Segment register information:

  Segment 0 base  (grows positive) : 0x00000010 (decimal 16)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x0000001e (decimal 30)
  Segment 1 limit                  : 2

Virtual Address Trace
  VA  0: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x0000001c (decimal:   28)
  VA  1: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x0000001c (decimal:   28)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
```



첫 번째 케이스를 그림으로 나타내면 위와 같다. Virtual address space는 0부터 시작하고 문제에서는 2개의 segment를 생성하였다. 또한, 하나의 segment는 grow negative이므로 그림의 왼쪽처럼 그릴 수 있다. Virtual address space는 0부터 시작하고 문제에서는 2개의 segment를 생성하였

다. 또한, 하나의 segment는 grow negative이므로 그림의 왼쪽처럼 도식화 할 수 있다. Virtual address를 physical address로 변환하기 위해서는 offset과 base of segment를 더하면 된다. 또한, offset은 virtual address에서 segment start address의 차로 구할 수 있다. 따라서 아래와 같은 결과가 도출된다.

- Virtual address : 14 => offset = -2 (seg 1) => physical address = -2+30 = 28
- Virtual address : 10 => offset = -6 (seg 1) => physical address = -6+30 = 24 => violation (seg1)
- Virtual address : 12 => offset = -4 (seg 1) => physical address = -4+30 = 26 => violation (seg1)
- Virtual address : 4 => offset = 4 (seg 0) => physical address = 4+16 = 20 => violation (seg0)

두 번째 케이스는 address space size가 32이고 physical memory size가 128이다. 이를 코드로 실행하고 그림으로 나타낸 결과는 아래와 같다. 나머지 태그 값은 첫 번째 케이스와 동일하다.

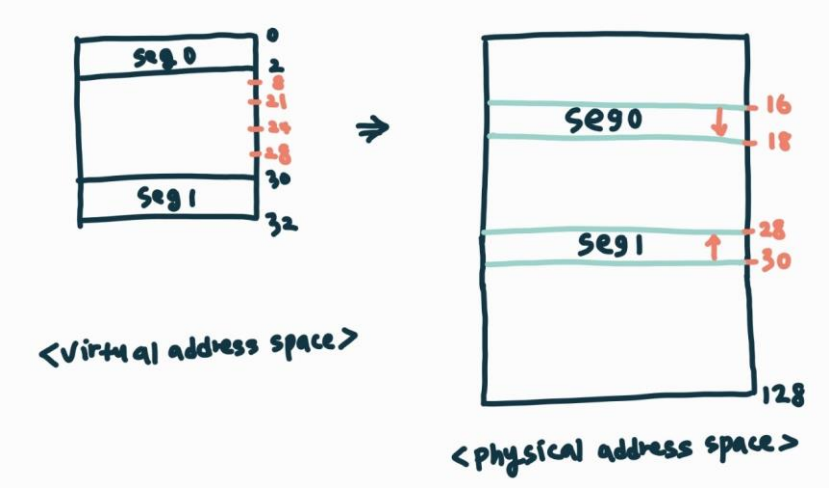
```
C:\Users\이나은\Downloads\OS_HW_2>python segmentation.py -s 20 -a 32 -p 128 -b 16 -l 2 -B 30 -L 2 -c
ARG seed 20
ARG address space size 32
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x00000010 (decimal 16)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x0000001e (decimal 30)
  Segment 1 limit                  : 2

Virtual Address Trace
  VA  0: 0x0000001c (decimal:   28) --> SEGMENTATION VIOLATION (SEG1)
  VA  1: 0x00000015 (decimal:   21) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x00000018 (decimal:   24) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x0000001c (decimal:   28) --> SEGMENTATION VIOLATION (SEG1)
  VA  4: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG0)
```



여기서 위와 같은 방식으로 translation을 진행할 수 있다.

- Virtual address : 8 => offset = 8 (seg 0) => physical address = 8+16 = 24 => violation (seg0)
- Virtual address : 21 => offset = -11 (seg 1) => physical address = -11+30 = 19 => violation (seg1)
- Virtual address : 24 => offset = -8 (seg 1) => physical address = -8+30 = 22 => violation (seg1)
- Virtual address : 28 => offset = -4 (seg 1) => physical address = -4+30 = 26 => violation (seg1)

위의 두 가지 케이스를 통해 각 segment가 동일한 base/limit register value를 가질 때 physical memory size / address space size (비율)의 값이 클수록 violation이 발생할 가능성이 크다는 것을 알 수 있다.

2) Different base and limit register values

위의 configuration을 충족하기 위해 -b, -l, -B, -L의 태그 값만 달리하여 실험을 수행하였다. 우선 첫 번째 케이스에서 base, limit register는 아래의 두 그림과 같다.

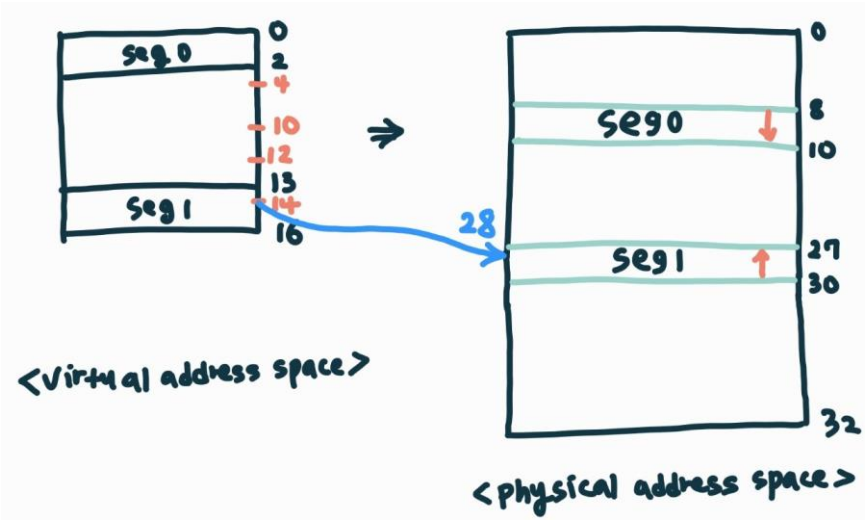
```
C:\Users\이나은\Downloads\OS_HW_2>python segmentation.py -s 20 -a 16 -p 32 -b 8 -l 2 -B 30 -L 3 -c
ARG seed 20
ARG address space size 16
ARG phys mem size 32

Segment register information:

  Segment 0 base  (grows positive) : 0x00000008 (decimal 8)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x0000001e (decimal 30)
  Segment 1 limit                  : 3

Virtual Address Trace
  VA  0: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x0000001c (decimal:   28)
  VA  1: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x0000001c (decimal:   28)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
```



- Virtual address : 14 => offset = -2 (seg 1) => physical address = -2+30 = 28
- Virtual address : 10 => offset = -6 (seg 1) => physical address = -6+30 = 24 => violation (seg1)
- Virtual address : 12 => offset = -4 (seg 1) => physical address = -4+30 = 26 => violation (seg1)
- Virtual address : 4 => offset = 4 (seg 0) => physical address = 4+8 = 12 => violation (seg0)

두 번째 케이스는 아래의 두 그림과 같다.

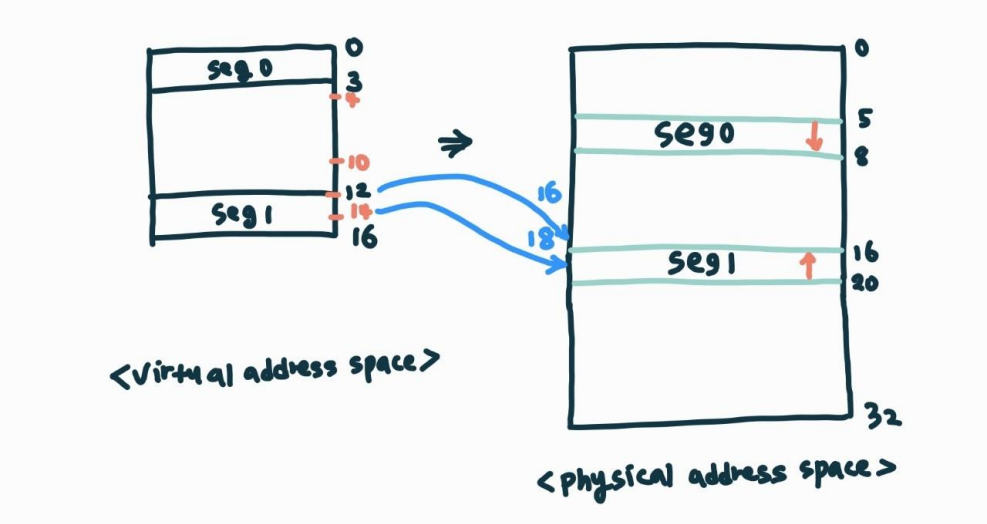
```
C:\Users\이나은\Downloads\OS_HW_2>python segmentation.py -s 20 -a 16 -p 32 -b 5 -l 3 -B 20 -L 4 -c
ARG seed 20
ARG address space size 16
ARG phys mem size 32

Segment register information:

  Segment 0 base  (grows positive) : 0x00000005 (decimal 5)
  Segment 0 limit                  : 3

  Segment 1 base  (grows negative) : 0x00000014 (decimal 20)
  Segment 1 limit                  : 4

Virtual Address Trace
  VA  0: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x00000012 (decimal:   18)
  VA  1: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x00000010 (decimal:   16)
  VA  3: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x00000012 (decimal:   18)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
```



- Virtual address : 14 => offset = -2 (seg 1) => physical address = -2+20 = 18
- Virtual address : 10 => offset = -6 (seg 1) => physical address = -6+20 = 14 => violation (seg1)
- Virtual address : 12 => offset = -4 (seg 1) => physical address = -4+20 = 16
- Virtual address : 4 => offset = 4 (seg 0) => physical address = 4+5 = 9 => violation (seg0)

위의 두 가지 케이스를 통해 동일한 address space size와 physical memory size를 가질 때 각 segment의 크기가 클수록 violation이 발생할 가능성이 낮다는 것을 알 수 있다.

#2. Paging – Linear Page Table

1) Address Translation

a) Random seed : 10, address space size : 16, physical memory size : 32, page size : 4

```
C:\Users\이나은\Downloads\OS_HW_2>python paging-linear-translate.py -s 10 -a 16 -p 32 -P 4 -c
ARG seed 10
ARG address space size 16
ARG phys mem size 32
ARG page size 4
ARG verbose False
ARG addresses -1
```

```
Page Table (from entry 0 down to the max size)
  0x80000003
  0x80000001
  0x80000006
  0x80000002

Virtual Address Trace
  VA 0x00000003 (decimal:        3) --> 0000000f (decimal       15) [VPN 0]
  VA 0x0000000f (decimal:       15) --> 0000000b (decimal       11) [VPN 3]
  VA 0x0000000f (decimal:       15) --> 0000000b (decimal       11) [VPN 3]
  VA 0x00000000 (decimal:        0) --> 0000000c (decimal       12) [VPN 0]
  VA 0x0000000d (decimal:       13) --> 00000009 (decimal        9) [VPN 3]
```

Page가 4개 존재하므로 4 bits의 virtual address 중 2개의 bit가 VPN을 나타내며 나머지 2 bits가 offset을 나타낸다. 또한, 5 bits의 physical memory address 중 3 bit가 PFN을 나타낸다. Physical address는 <PFN, offset>으로 나타낼 수 있다.

1. Virtual address 3 : 0011(2) -> VPN 00, offset 11

=> 0x80000003 : 10000000000000000000000000000011(2) -> PFN 011

=> Physical address 15 : 01111(2)

2. Virtual address 15 : 1111(2) -> VPN 11, offset 11

=> 0x80000002 : 10000000000000000000000000000010(2) -> PFN 010

=> Physical address 11 : 01011(2)

3. Virtual address 0 : 0000(2) -> VPN 00, offset 00

=> 0x80000003 : 10000000000000000000000000000011(2) -> PFN 011

=> Physical address 12 : 01100(2)

4. Virtual address 13 : 1101(2) -> VPN 11, offset 01

=> 0x80000002 : 10000000000000000000000000000010(2) -> PFN 010

=> Physical address 9 : 01001(2)

b) Random seed : 20, address space size : 32, physical memory size : 64, page size : 2

```
C:\Users\이나은\Downloads\OS_HW_2>python paging-linear-translate.py -s 20 -a 32 -p 64 -P 2 -c
ARG seed 20
ARG address space size 32
ARG phys mem size 64
ARG page size 2
ARG verbose False
ARG addresses -1
```

```
Page Table (from entry 0 down to the max size)
  0x80000015
  0x8000001c
  0x00000000
  0x80000012
  0x00000000
  0x00000000
  0x80000003
  0x00000000
  0x8000000e
  0x00000000
  0x00000000
  0x8000000a
  0x00000000
  0x80000019
  0x00000000
  0x00000000

Virtual Address Trace
  VA 0x00000012 (decimal:       18) -->  Invalid (VPN 9 not valid)
  VA 0x00000001 (decimal:        1) --> 0000002b (decimal       43) [VPN 0]
  VA 0x00000003 (decimal:        3) --> 00000039 (decimal       57) [VPN 1]
  VA 0x00000006 (decimal:        6) --> 00000024 (decimal       36) [VPN 3]
  VA 0x00000016 (decimal:       22) --> 00000014 (decimal       20) [VPN 11]
```

Page가 16개 존재하므로 5 bits의 virtual address 중 4개의 bit가 VPN을 나타내며 나머지 1 bit가 offset을 나타낸다. 또한, 6 bits의 physical memory address 중 5 bit가 PFN을 나타낸다. Physical

address는 <PFN, offset>으로 나타낼 수 있다.

1. Virtual address 18 : 10010(2) -> VPN 1001, offset 0

=> 0x00000000

=> 해당 페이지가 invalid 하므로 physical address로 변환할 수 없다.

2. Virtual address 1 : 00001(2) -> VPN 0000, offset 1

=> 0x80000015 : 10000000000000000000000000010101(2) -> PFN : 10101

=> Physical address 43 : 101011 (2)

3. Virtual address 3 : 00011(2) -> VPN 0001, offset 1

=> 0x8000001c : 10000000000000000000000000011100(2) -> PFN : 11100

=> Physical address 57 : 111001(2)

4. Virtual address 6 : 00110(2) -> VPN 0011, offset 0

=> 0x80000012 : 10000000000000000000000000010010(2) -> PFN : 10010

=> Physical address 36 : 100100

5. Virtual address 22 : 10110(2) -> VPN 1011, offset 0

=> 0x8000000a : 10000000000000000000000000001010(2) -> PFN : 01010

=> Physical address 20 : 010100

c) Random seed : 30, address space size : 64, physical memory size : 256, page size : 4

```
C:\Users\이나은\Downloads\OS_HW_2>python paging-linear-translate.py -s 30 -a 64 -p 256 -P 8 -c
ARG seed 30
ARG address space size 64
ARG phys mem size 256
ARG page size 8
ARG verbose False
ARG addresses -1
```

```
Page Table (from entry 0 down to the max size)
  0x80000009
  0x00000000
  0x80000006
  0x00000000
  0x00000000
  0x8000001f
  0x00000000
  0x00000000

Virtual Address Trace
  VA 0x00000004 (decimal:        4) --> 0000004c (decimal       76) [VPN 0]
  VA 0x0000000a (decimal:       10) -->  Invalid (VPN 1 not valid)
  VA 0x00000035 (decimal:       53) -->  Invalid (VPN 6 not valid)
  VA 0x00000026 (decimal:       38) -->  Invalid (VPN 4 not valid)
  VA 0x0000003a (decimal:       58) -->  Invalid (VPN 7 not valid)
```

Page가 8개 존재하므로 6 bits의 virtual address 중 3개의 bit가 VPN을 나타내며 나머지 3 bits가 offset을 나타낸다. 또한, 8 bits의 physical memory address 중 5 bit가 PFN을 나타낸다. Physical address는 <PFN, offset>으로 나타낼 수 있다.

1. Virtual address 4 : 000100(2) -> VPN 000, offset 100

=> 0x80000009 : 10000000000000000000000000001001(2) – PFN 01001

=> Physical address 76 : 001001100

2. Virtual address 10 : 001010(2) -> VPN 001, offset 010

=> 0x00000000

=> 해당 페이지가 invalid 하므로 physical address로 변환할 수 없다.

3. Virtual address 53 : 110101(2) -> VPN 110, offset 101

=> 0x00000000

=> 해당 페이지가 invalid 하므로 physical address로 변환할 수 없다.

4. Virtual address 38 : 100110(2) -> VPN 100, offset 110

=> 0x00000000

=> 해당 페이지가 invalid 하므로 physical address로 변환할 수 없다.

5. Virtual address 58 : 10110(2) -> VPN 1011, offset 0

=> 0x00000000

=> 해당 페이지가 invalid 하므로 physical address로 변환할 수 없다.

2) Page table size

Page table size는 page table entry의 개수와 각 entry의 size에 의해 결정된다. 해당 코드의 옵션 중 address space size와 page size를 통해 VPN을 결정할 수 있고 이를 통해 entry의 개수를 결정할 수 있다. Address space size가 증가할수록 entry의 size가 증가하고 address space size가 크고 page size가 작을 때 entry의 개수가 증가한다.

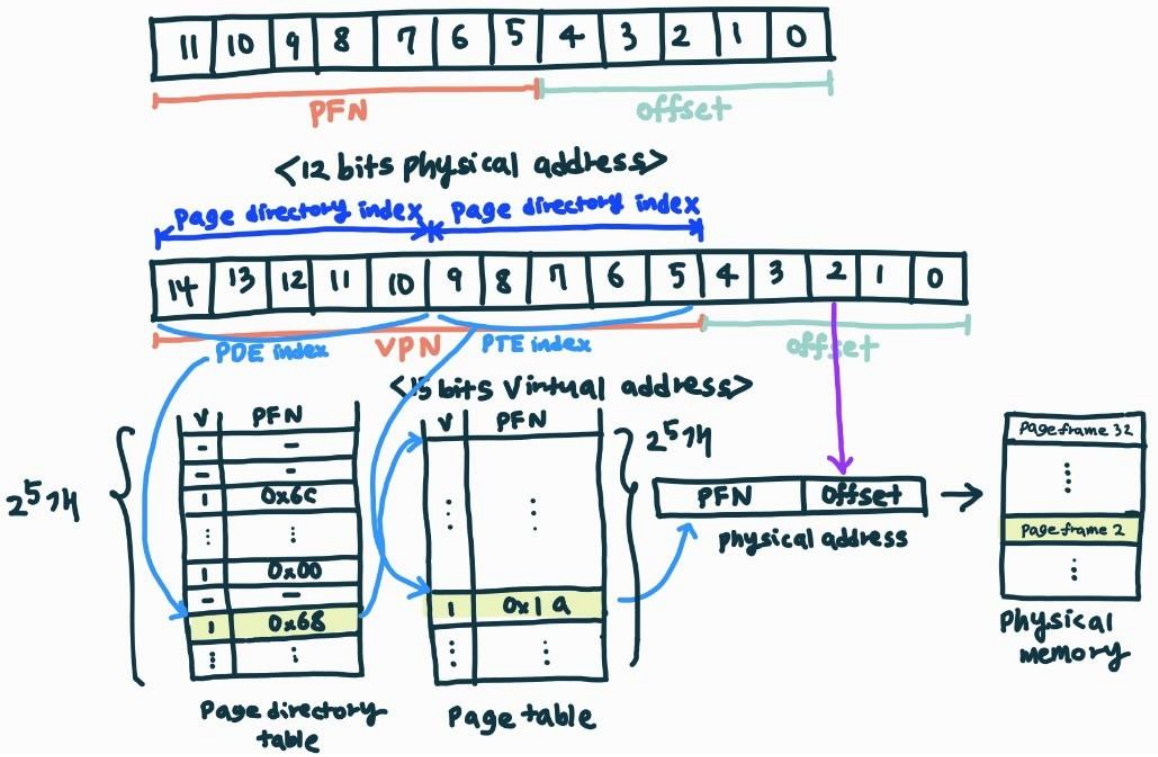
#3. Paging: Multi-Level Page Tables

<physical memory>
Page size : 2^5
# of pages : 2^7 → PFN : 7 bits
offset : 5 bits

* 노란색 밑줄
: 코드에 정의되어 있는 값

<virtual address>
Page size : 2^5
# of pages : 2^{10} → VPN : 10 bits
offset : 5 bits

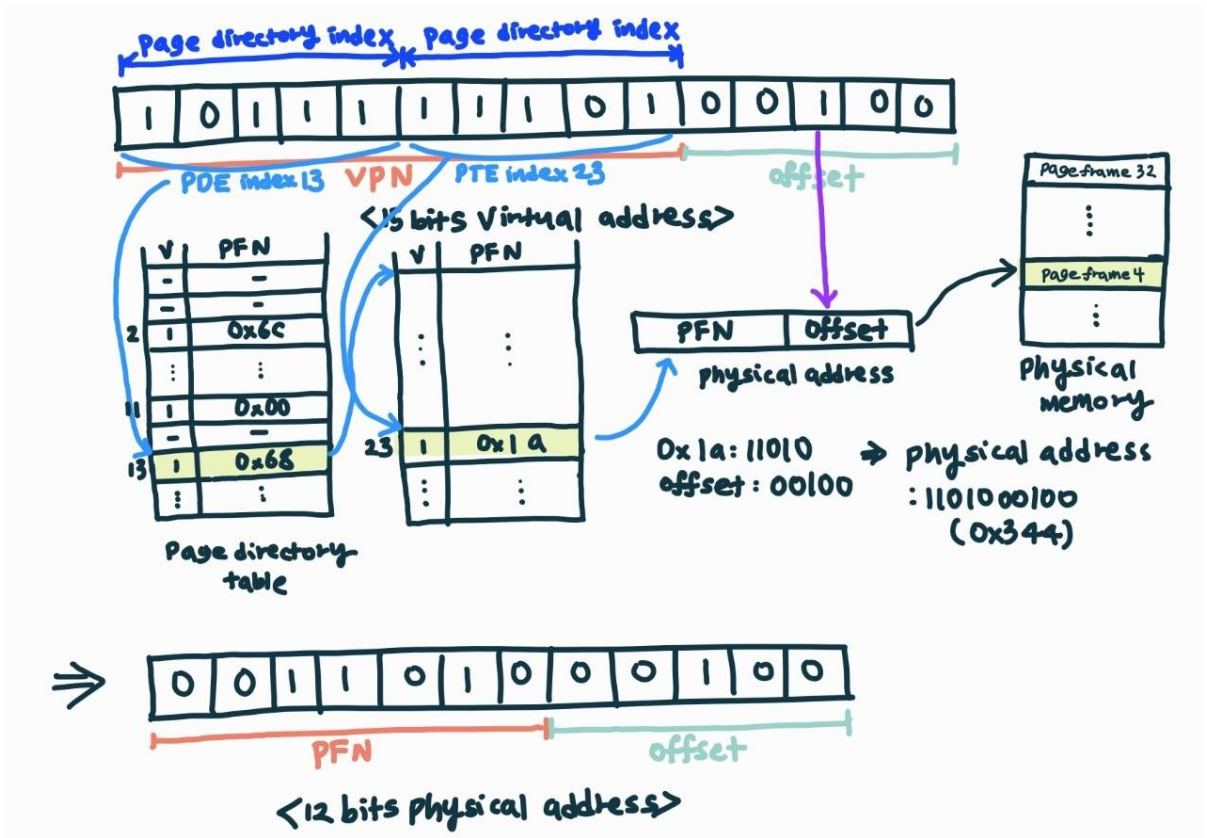
⇒ # of page table entries : 2^{10}
size of linear page table : $2^5 \times 2^{10} = 2^{15}$
page당 table entry 개수 : $2^{10}/2^5 = 2^5$
size of PDT : $2^{10} \times 1 = 2^{10}$
∴ 1 PDT is stored in 2^5 frames



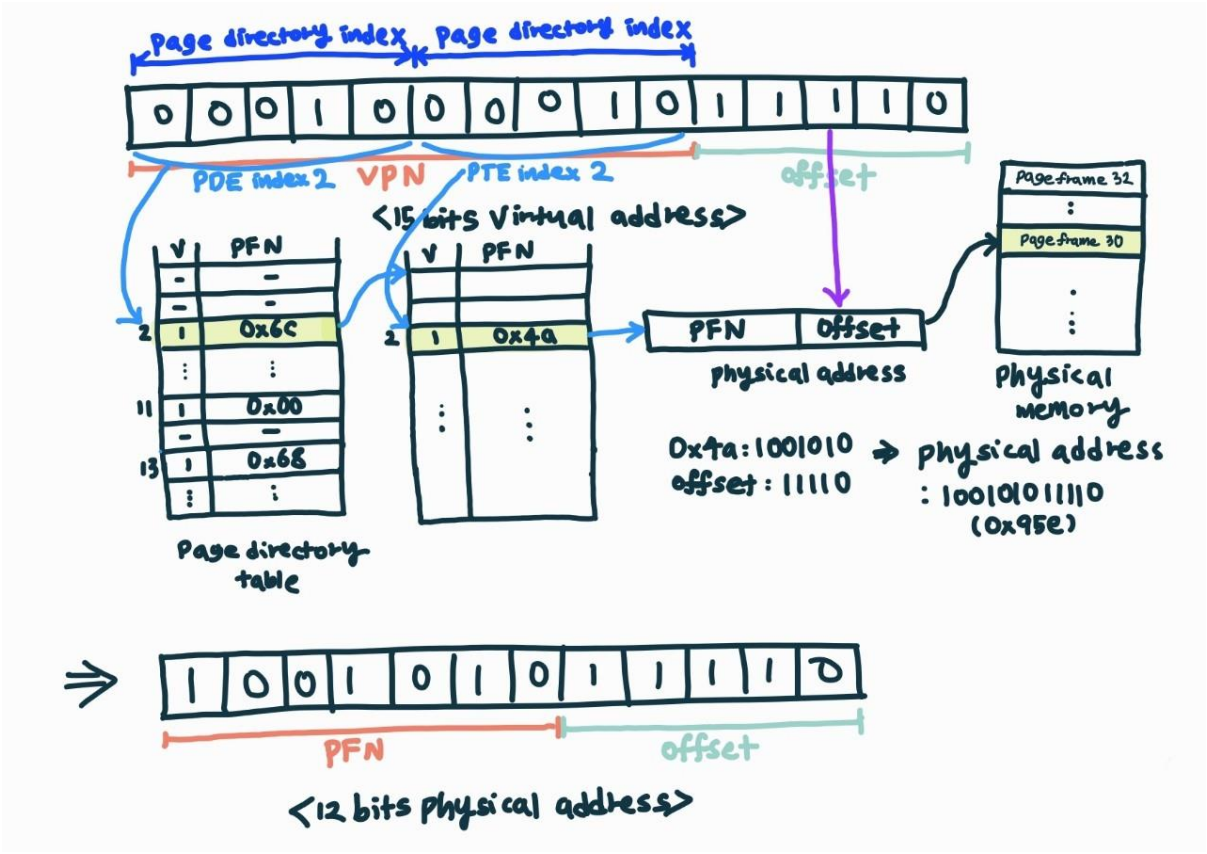
```
PDBR: 73  (decimal) [This means the page directory is held in this page]

Virtual Address 36e4:
  --> pde index:0xd [decimal 13] pde contents:0xe8 (valid 1, pfn 0x68 [decimal 104])
    --> pte index:0x17 [decimal 23] pte contents:0x9a (valid 1, pfn 0x1a [decimal 26])
      --> Translates to Physical Address 0x344 --> Value: 0a
Virtual Address 085e:
  --> pde index:0x2 [decimal 2] pde contents:0xec (valid 1, pfn 0x6c [decimal 108])
    --> pte index:0x2 [decimal 2] pte contents:0xca (valid 1, pfn 0x4a [decimal 74])
      --> Translates to Physical Address 0x95e --> Value: 18
Virtual Address 2c75:
  --> pde index:0xb [decimal 11] pde contents:0x80 (valid 1, pfn 0x00 [decimal 0])
    --> pte index:0x3 [decimal 3] pte contents:0xb7 (valid 1, pfn 0x37 [decimal 55])
      --> Translates to Physical Address 0x6f5 --> Value: 0f
```

1) Virtual address : 36e4 (101111110100100)



2) Virtual address : 085e (000100001011110)



3) Virtual address : 2c75 (010110001110101)