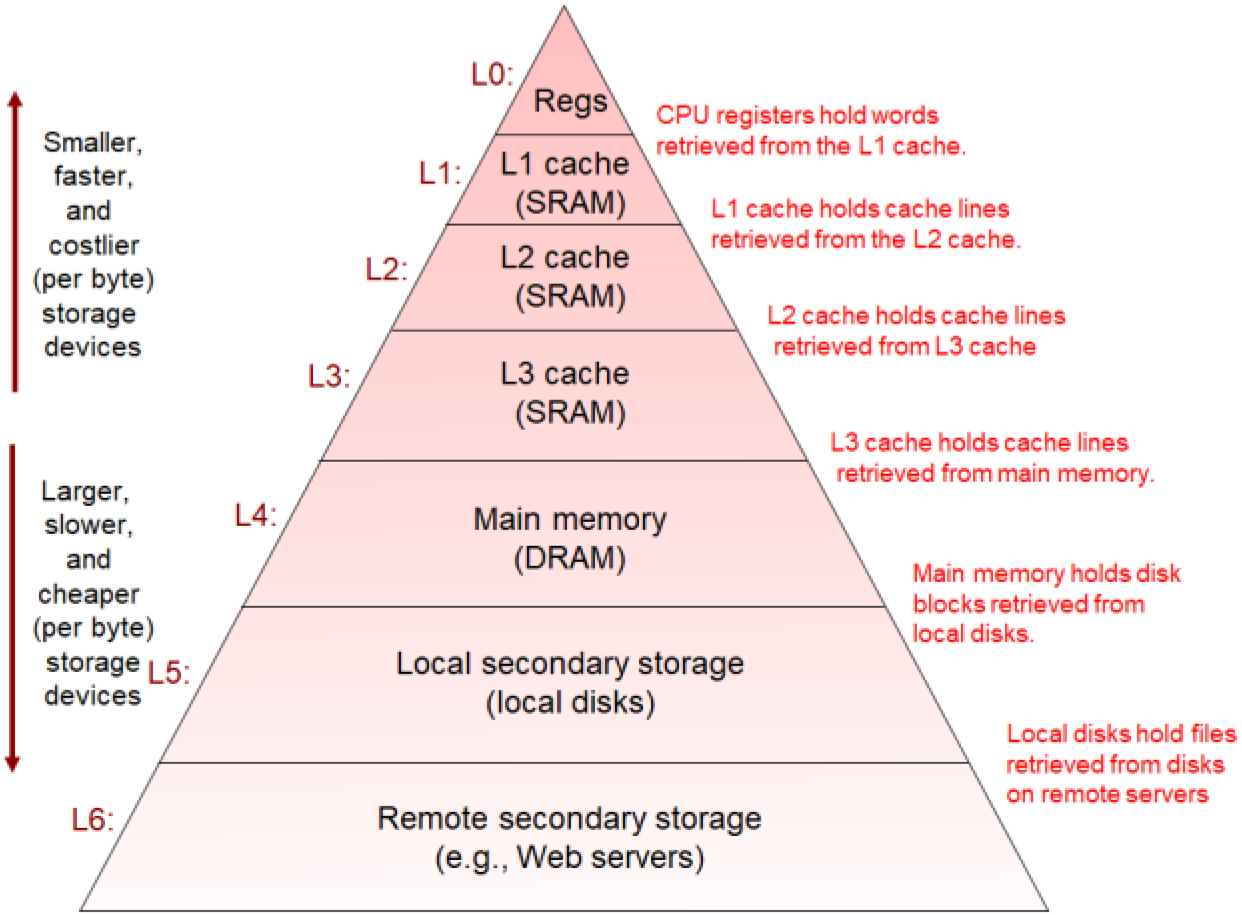
2nd project

20102119 임보영
20102122 정효안
20102123 최진아

CONTENTS

1. Memory Hierarchy
2. Cache Memory
3. Design Memory Hierarchy

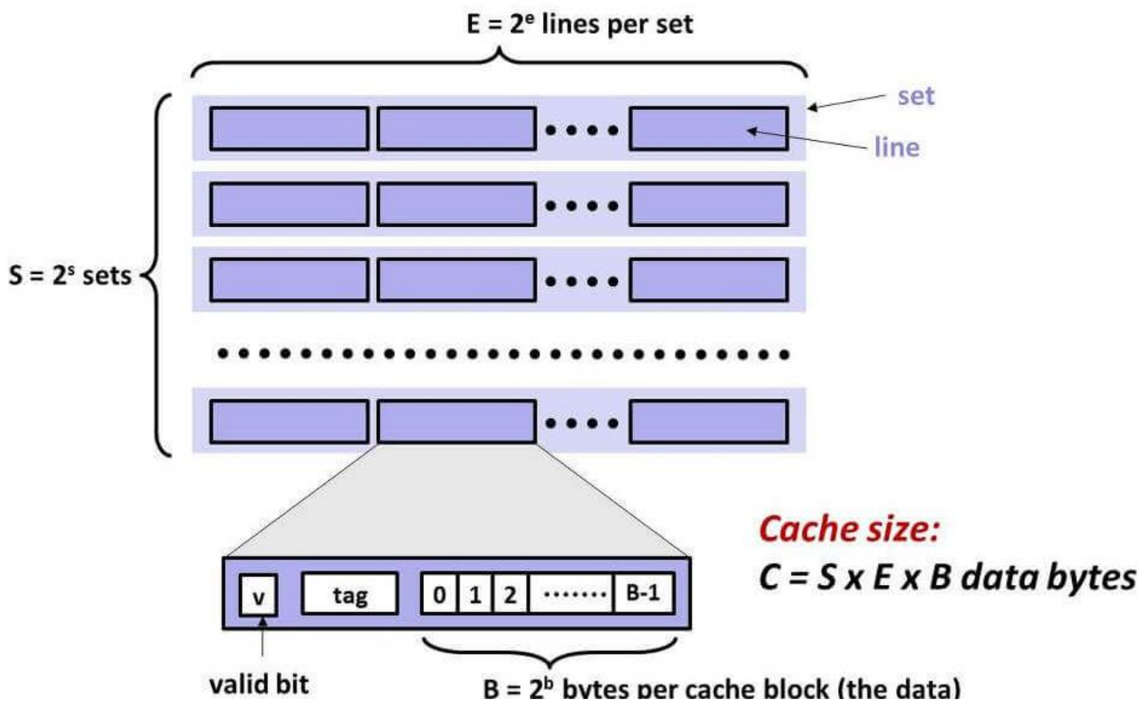
01. Memory hierarchy


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (e.g., Web servers)
CPU registers hold words retrieved from the L1 cache.
L1 cache holds cache lines retrieved from the L2 cache.
L2 cache holds cache lines retrieved from L3 cache
L3 cache holds cache lines retrieved from main memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote servers

- **L1 : 1 element**
- **L2 : 16 element**
- **L3 : 256 element**
- **L4 : 4096 element**

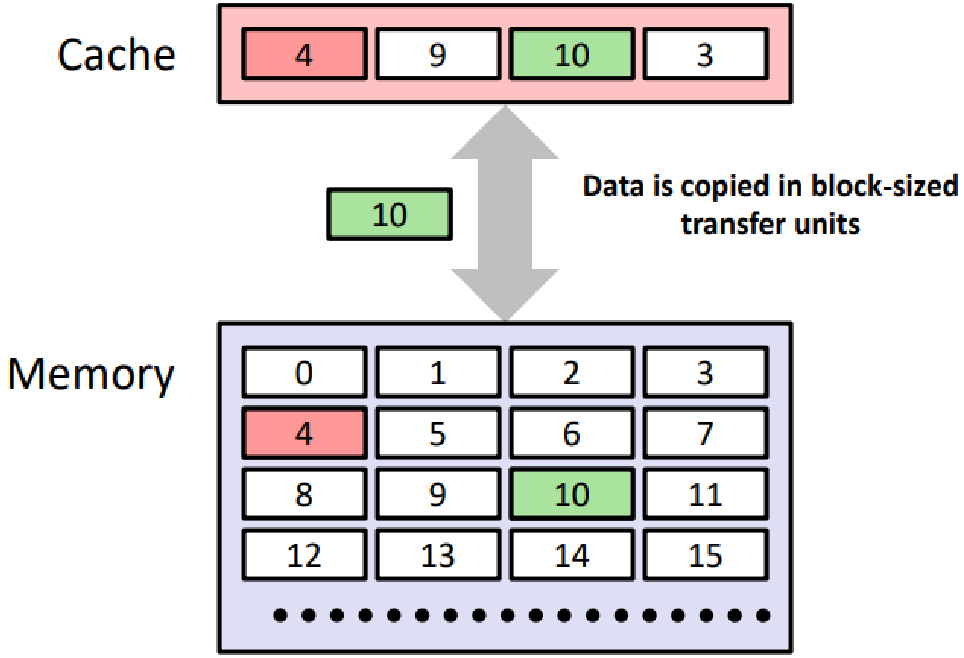
02. Cache Memory

General Cache Organization (S, E, B)



Cache size:
C = S x E x B data bytes

- Cache Organization



Data is copied in block-sized transfer units

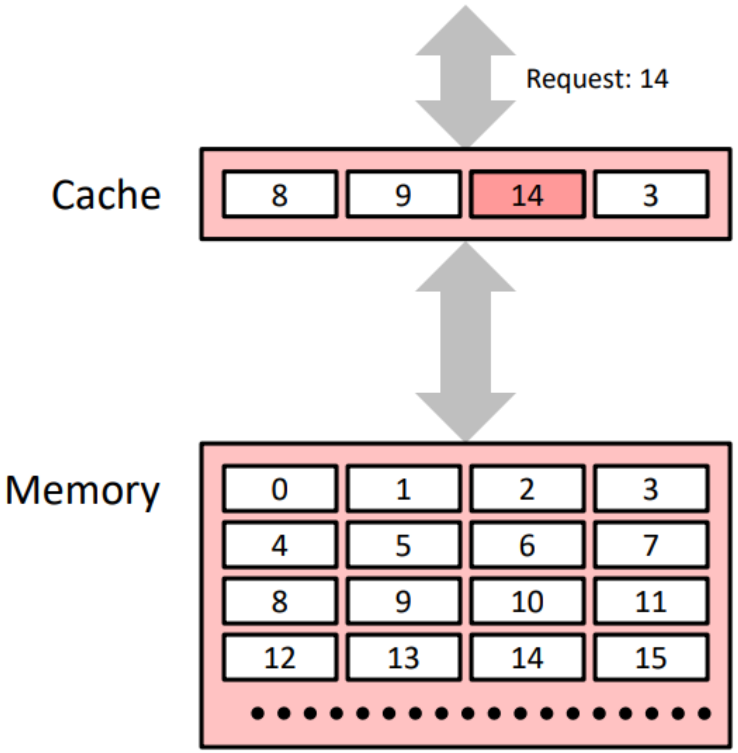
- Cache ↔ Main Memory

02. Cache Memory

- **Hit & Miss**
  - **Data in Block b is needed**

    → if block b is in cache :

    Hit!



02. Cache Memory

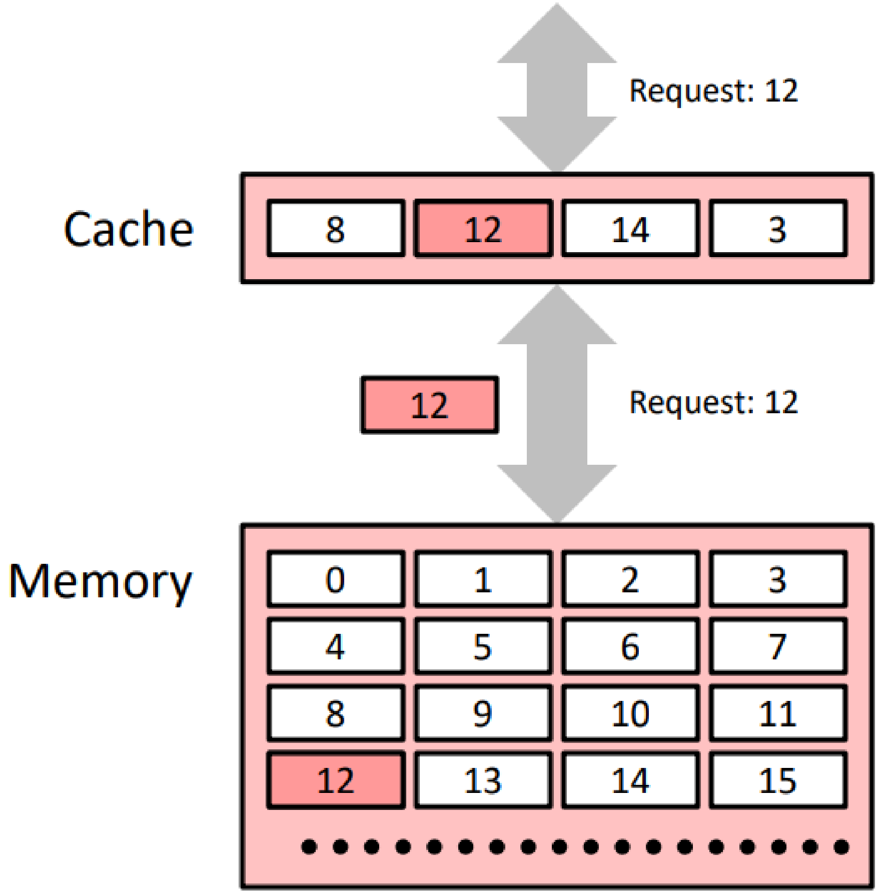
- **Hit & Miss**
  - **Data in Block b is needed**

    → if block b is not in cache :

Miss!

Block : memory → Cache

Fetched



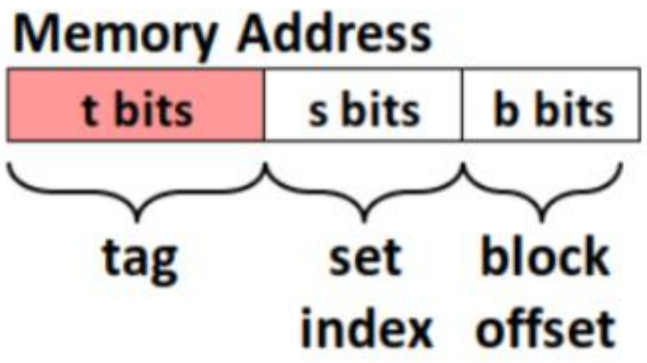
Cache

8	12	14	3

Memory

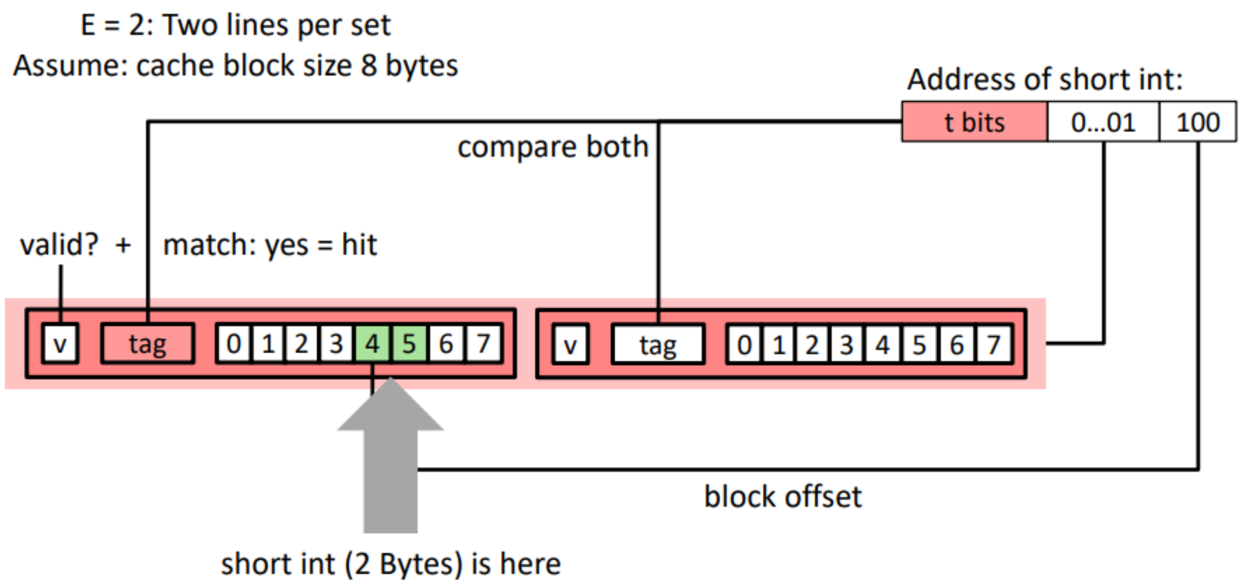
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

02. Cache Memory

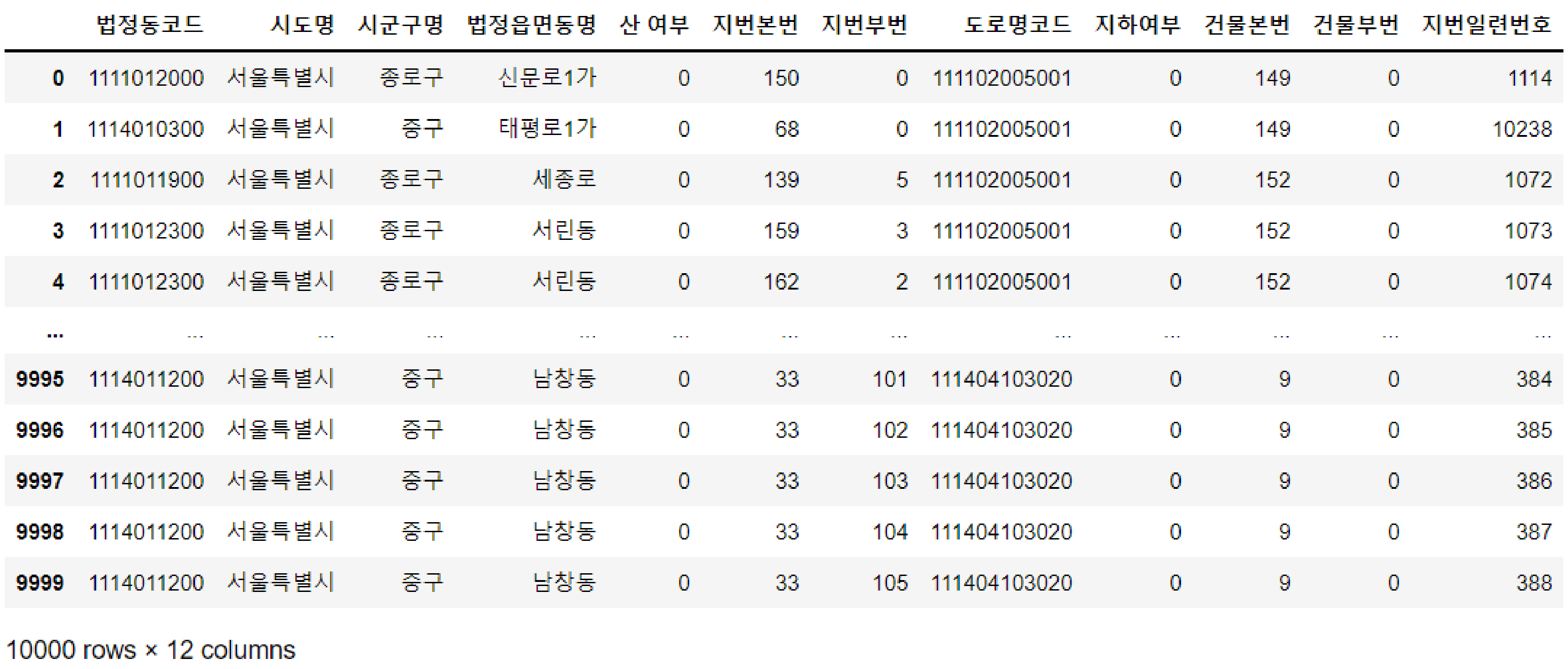


- L1 : 1 element
- L2 : 16 element
- L3 : 256 element
- L4 : 4096 element

2-way set associate cache



+ Real-world data set

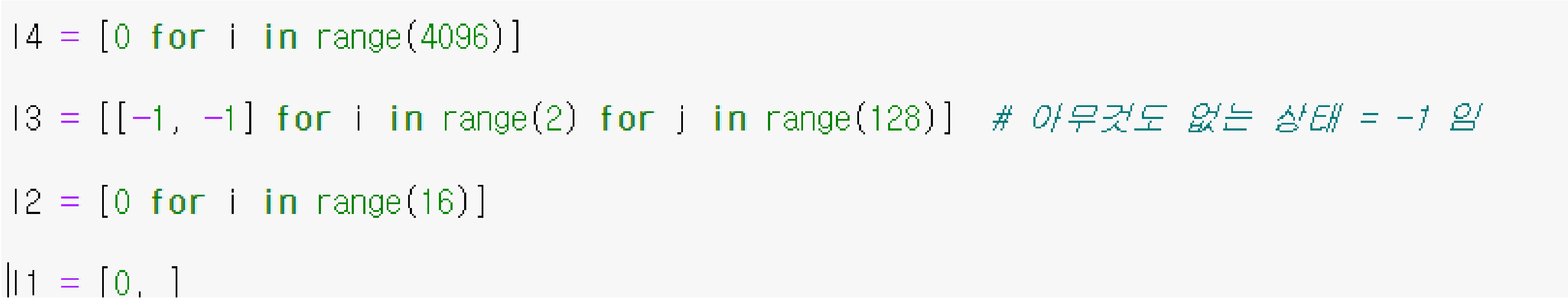


	법정동코드	시도명	시군구명	법정읍면동명	산 여부	지번본번	지번부번	도로명코드	지하여부	건물본번	건물부번	지번일련번호
0	1111012000	서울특별시	종로구	신문로1가	0	150	0	111102005001	0	149	0	1114
1	1114010300	서울특별시	중구	태평로1가	0	68	0	111102005001	0	149	0	10238
2	1111011900	서울특별시	종로구	세종로	0	139	5	111102005001	0	152	0	1072
3	1111012300	서울특별시	종로구	서린동	0	159	3	111102005001	0	152	0	1073
4	1111012300	서울특별시	종로구	서린동	0	162	2	111102005001	0	152	0	1074
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	1114011200	서울특별시	중구	남창동	0	33	101	111404103020	0	9	0	384
9996	1114011200	서울특별시	중구	남창동	0	33	102	111404103020	0	9	0	385
9997	1114011200	서울특별시	중구	남창동	0	33	103	111404103020	0	9	0	386
9998	1114011200	서울특별시	중구	남창동	0	33	104	111404103020	0	9	0	387
9999	1114011200	서울특별시	중구	남창동	0	33	105	111404103020	0	9	0	388

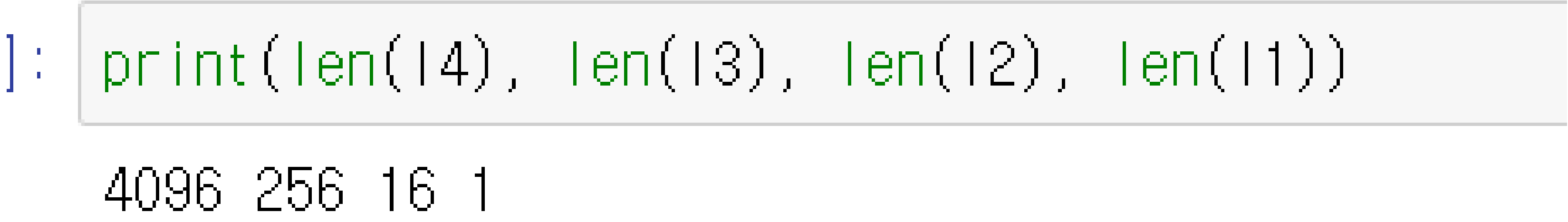
10000 rows × 12 columns

- Road base data – building DB (한국지역정보개발원 도로명주소 DB)
- Data : 10000

03. Design memory hierarchy



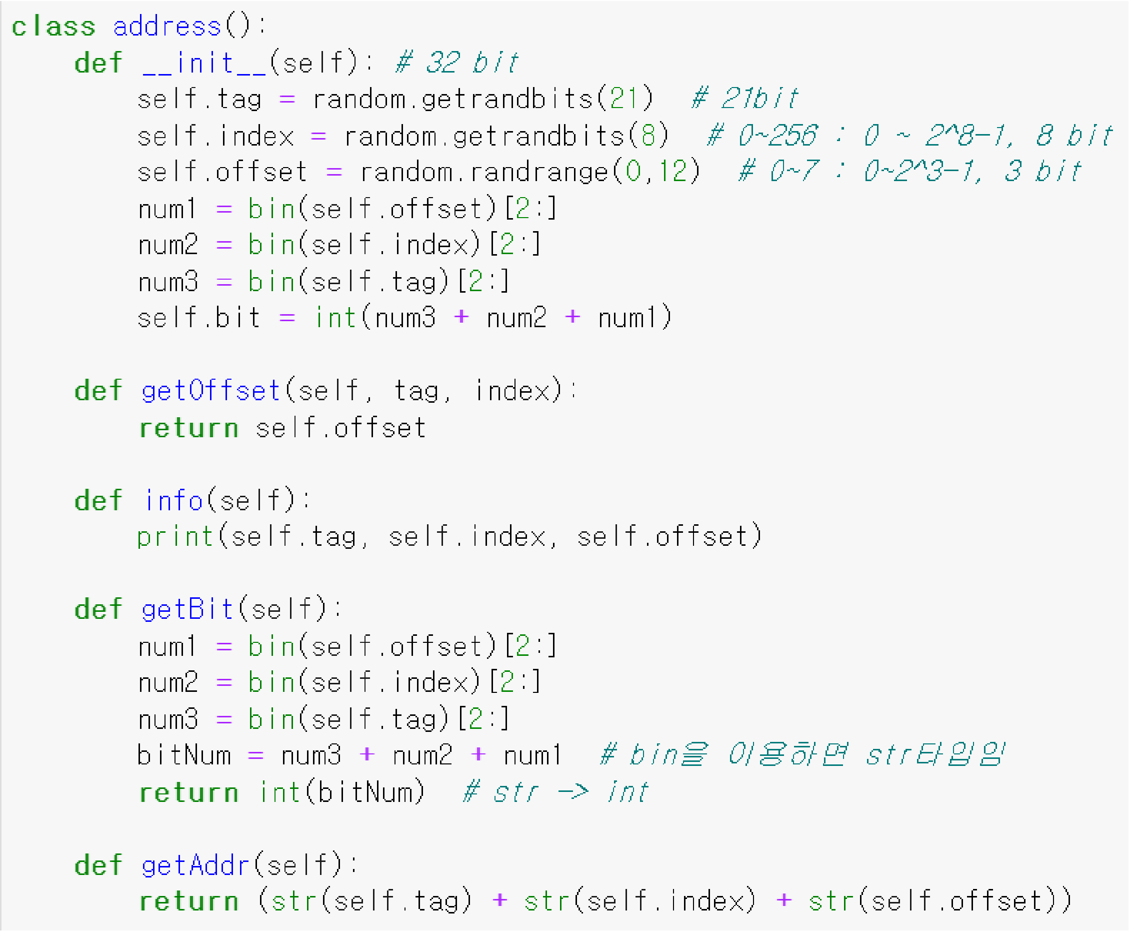
```
l4 = [0 for i in range(4096)]

l3 = [[-1, -1] for i in range(2) for j in range(128)]  # 아무것도 없는 상태 = -1 임

l2 = [0 for i in range(16)]

l1 = [0, ]
```



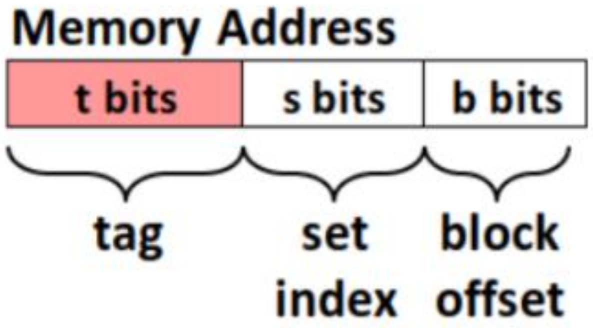
```
]: print(len(l4), len(l3), len(l2), len(l1))

   4096 256 16 1
```

- Memory hierarchy’s layers : l1(1), l2(16), l3(256), l4(4096)
- L3 layer : 2-way set associate cache -> two-dimensional array (l3[128][2])

03. Design memory hierarchy



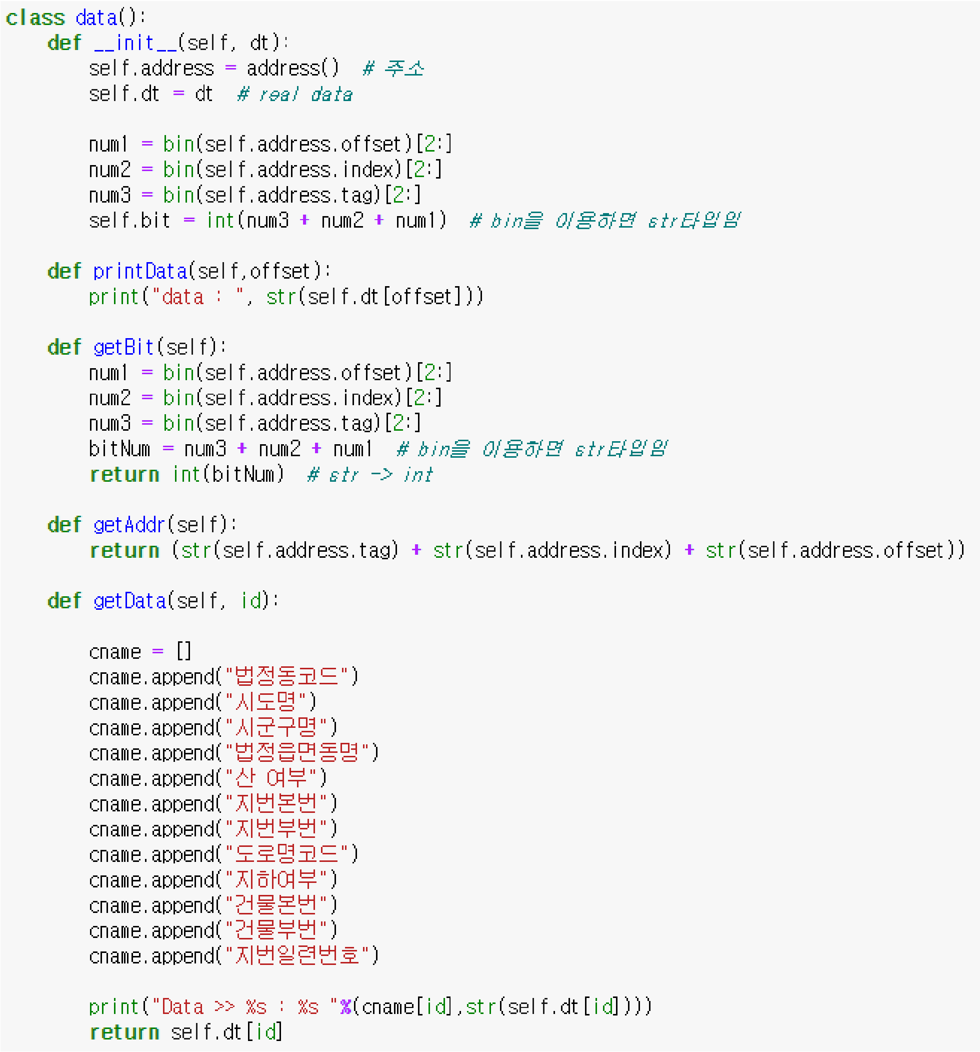
```
class address():
    def __init__(self): # 32 bit
        self.tag = random.getrandbits(21)  # 21bit
        self.index = random.getrandbits(8)  # 0~256 : 0 ~ 2^8-1, 8 bit
        self.offset = random.randrange(0,12)  # 0~7 : 0~2^3-1, 3 bit
        num1 = bin(self.offset)[2:]
        num2 = bin(self.index)[2:]
        num3 = bin(self.tag)[2:]
        self.bit = int(num3 + num2 + num1)

    def getOffset(self, tag, index):
        return self.offset

    def info(self):
        print(self.tag, self.index, self.offset)

    def getBit(self):
        num1 = bin(self.offset)[2:]
        num2 = bin(self.index)[2:]
        num3 = bin(self.tag)[2:]
        bitNum = num3 + num2 + num1  # bin을 이용하면 str타입임
        return int(bitNum)  # str -> int

    def getAddr(self):
        return (str(self.tag) + str(self.index) + str(self.offset))
```



- Address Class :
  - Address: 32 bit
  - Tag : 21 bit
  - set index: $\log_2 256 = 8$ bit
  - block offset : $\log_2 8 = 3$ bit

→ random value

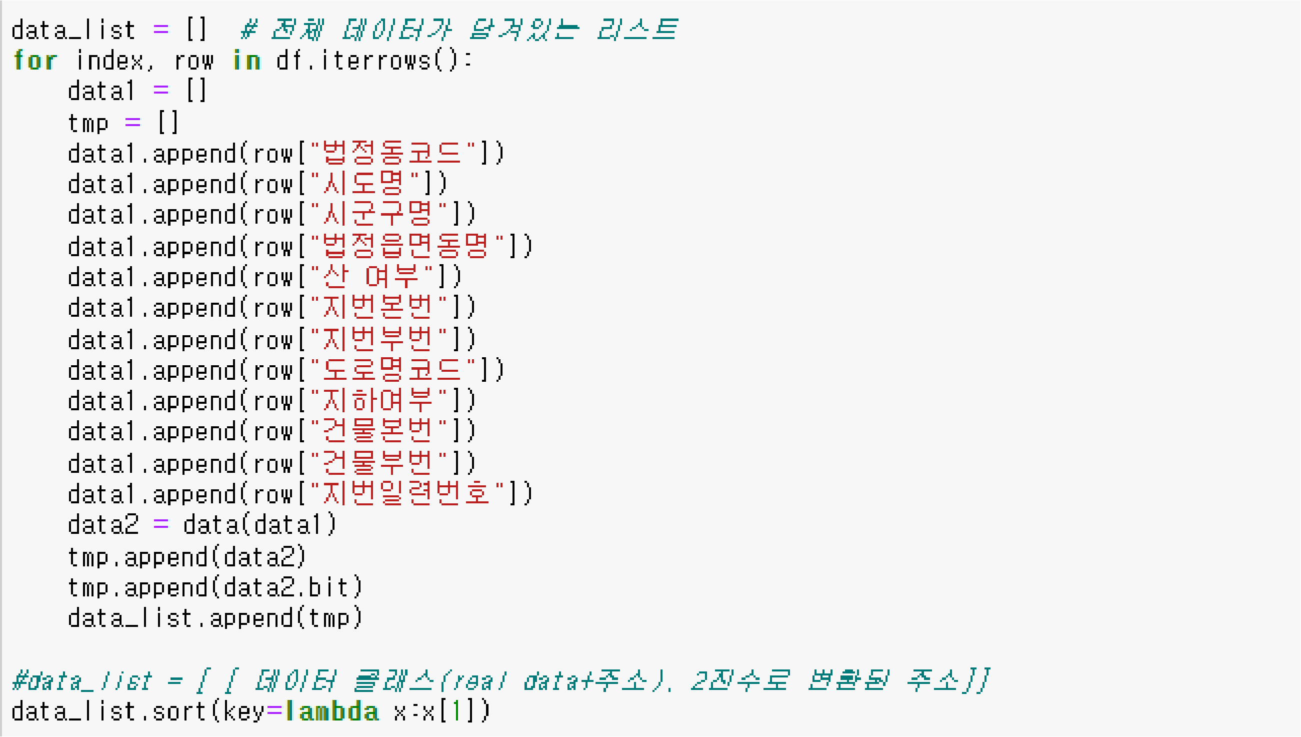
03. Design memory hierarchy



```
class data():
    def __init__(self, dt):
        self.address = address()  # 주소
        self.dt = dt  # real data

        num1 = bin(self.address.offset)[2:]
        num2 = bin(self.address.index)[2:]
        num3 = bin(self.address.tag)[2:]
        self.bit = int(num3 + num2 + num1)  # bin을 이용하면 str타입임

    def printData(self,offset):
        print("data : ", str(self.dt[offset]))

    def getBit(self):
        num1 = bin(self.address.offset)[2:]
        num2 = bin(self.address.index)[2:]
        num3 = bin(self.address.tag)[2:]
        bitNum = num3 + num2 + num1  # bin을 이용하면 str타입임
        return int(bitNum)  # str -> int

    def getAddr(self):
        return (str(self.address.tag) + str(self.address.index) + str(self.address.offset))

    def getData(self, id):

        cname = []
        cname.append("법정동코드")
        cname.append("시도명")
        cname.append("시군구명")
        cname.append("법정읍면동명")
        cname.append("산 여부")
        cname.append("지번본번")
        cname.append("지번부번")
        cname.append("도로명코드")
        cname.append("지하여부")
        cname.append("건물본번")
        cname.append("건물부번")
        cname.append("지번일련번호")

        print("Data >> %s : %s "%(cname[id],str(self.dt[id])))
        return self.dt[id]
```

- Data Class :
    - address + Real Data
    - Address will work as a key
    - “dt” includes all columns of datas
        - Array : offset will be the index of the array.

03. Design memory hierarchy



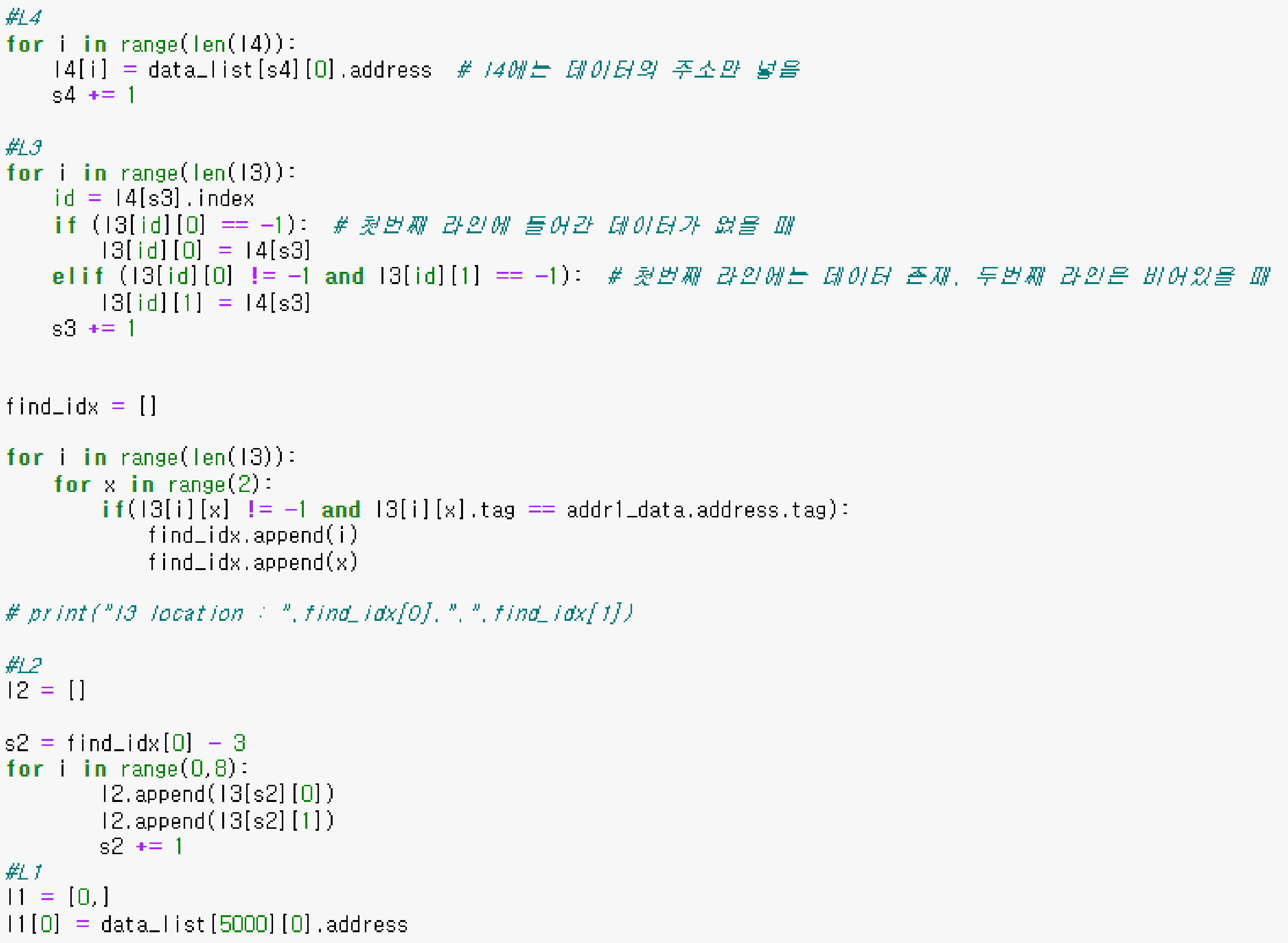
```
data_list = []  # 전체 데이터가 담겨있는 리스트
for index, row in df.iterrows():
    data1 = []
    tmp = []
    data1.append(row["법정동코드"])
    data1.append(row["시도명"])
    data1.append(row["시군구명"])
    data1.append(row["법정읍면동명"])
    data1.append(row["산 여부"])
    data1.append(row["지번본번"])
    data1.append(row["지번부번"])
    data1.append(row["도로명코드"])
    data1.append(row["지하여부"])
    data1.append(row["건물본번"])
    data1.append(row["건물부번"])
    data1.append(row["지번일련번호"])
    data2 = data(data1)
    tmp.append(data2)
    tmp.append(data2.bit)
    data_list.append(tmp)

#data_list = [ [ 데이터 클래스(real data+주소), 2진수로 변환된 주소]]
data_list.sort(key=lambda x:x[1])
```

- data_list : includes all real data.
  - Two-dimensional Array :

    real data + address
  - Data stored in the data_list are arranged in order of address.

03. Design memory hierarchy

- Data write



```
#L4
for i in range(len(l4)):
    l4[i] = data_list[s4][0].address  # l4에는 데이터의 주소만 넣음
    s4 += 1

#L3
for i in range(len(l3)):
    id = l4[s3].index
    if (l3[id][0] == -1):  # 첫번째 라인에 들어간 데이터가 없을 때
        l3[id][0] = l4[s3]
    elif (l3[id][0] != -1 and l3[id][1] == -1):  # 첫번째 라인에는 데이터 존재, 두번째 라인은 비어있을 때
        l3[id][1] = l4[s3]
    s3 += 1


find_idx = []

for i in range(len(l3)):
    for x in range(2):
        if(l3[i][x] != -1 and l3[i][x].tag == addr1_data.address.tag):
            find_idx.append(i)
            find_idx.append(x)

# print("l3 location : ",find_idx[0],",",find_idx[1])

#L2
l2 = []

s2 = find_idx[0] - 3
for i in range(0,8):
        l2.append(l3[s2][0])
        l2.append(l3[s2][1])
        s2 += 1
#L1
l1 = [0,]
l1[0] = data_list[5000][0].address
```

- Require address A

  11010111001111010101011001100111

  (address of data_list[5000])

  -> there is not A in cache.

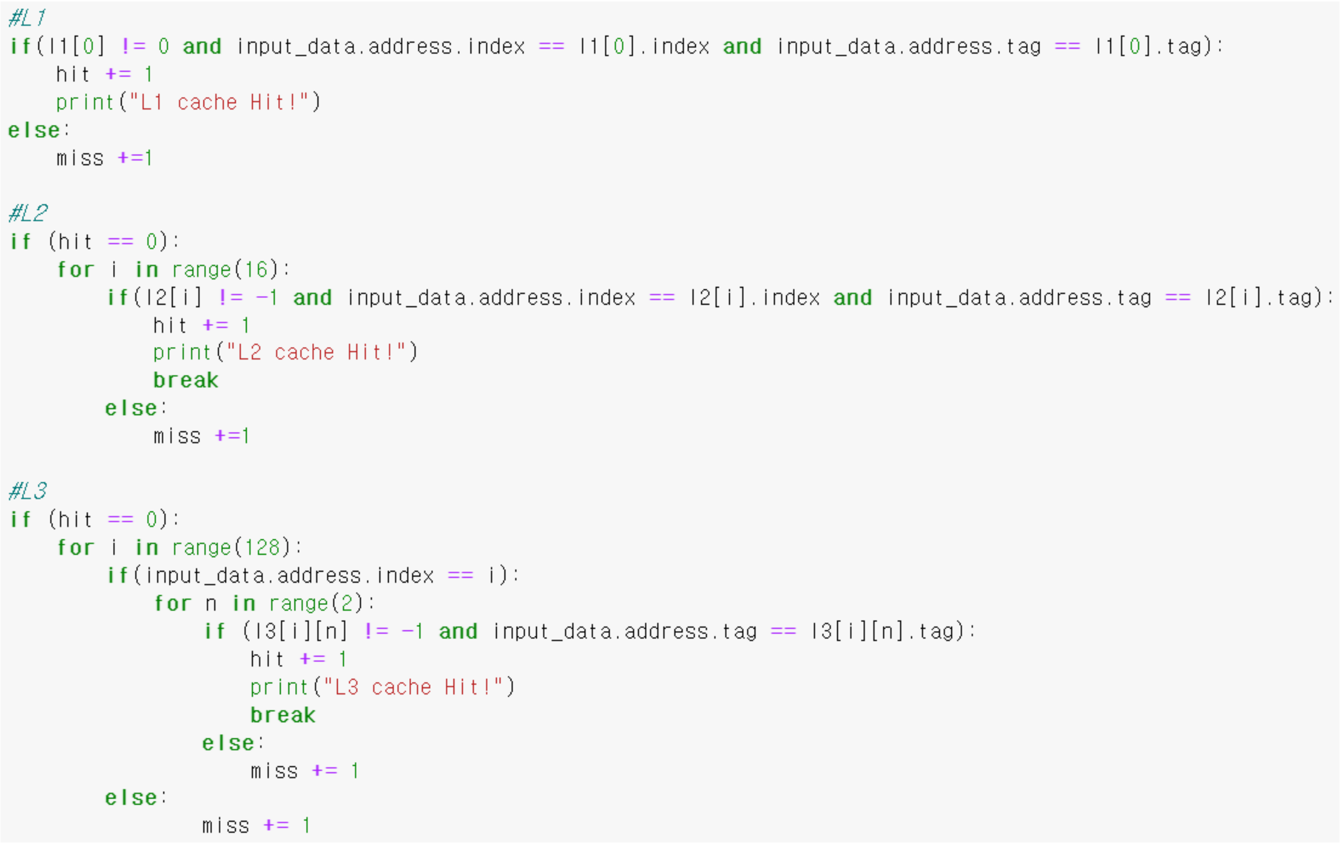
  -> The addresses adjacent to A will be stored with A (spatial locality)

- A will be fetched from memory.
- A is fetched with tag and index in l3 because E = 2.

03. Design memory hierarchy

- Data read

Require address 110101110011110101011001100111(address of data_list[5000])



```
#L1
if(l1[0] != 0 and input_data.address.index == l1[0].index and input_data.address.tag == l1[0].tag):
    hit += 1
    print("L1 cache Hit!")
else:
    miss +=1

#L2
if (hit == 0):
    for i in range(16):
        if(l2[i] != -1 and input_data.address.index == l2[i].index and input_data.address.tag == l2[i].tag):
            hit += 1
            print("L2 cache Hit!")
            break
        else:
            miss +=1

#L3
if (hit == 0):
    for i in range(128):
        if(input_data.address.index == i):
            for n in range(2):
                if (l3[i][n] != -1 and input_data.address.tag == l3[i][n].tag):
                    hit += 1
                    print("L3 cache Hit!")
                    break
                else:
                    miss += 1
        else:
            miss += 1
```

- L1 -> L2 -> L3 -> L4
- Compare tag
- In l3, it's 2-way
    - Set index determines the set and then, tag determines the line.
- If address in cache : hit!

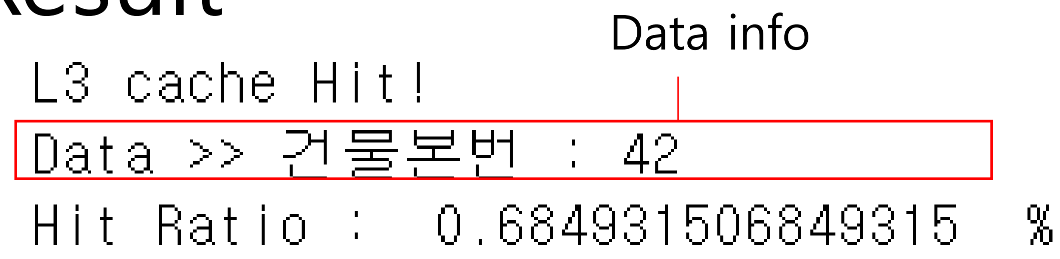
03. Design memory hierarchy

- Hit ratio
  - Hit ratio = hit / access
  - If address A is referenced, address B adjacent to A is likely to be referenced.

Therefore, after referring to address A, if referring to address B, it will be Hit.

  - Hit ratio is calculated in “read” code.
    - address A is 110101110011110101011001100111
    - Address B which is adjacent to A is 1101100010110000000101111000111

Result



```
L3 cache Hit!
Data >> 건물본번 : 42
Hit Ratio :  0.684931506849315  %
```

- B is in L3
- Hit ratio is about 0.68%
- Offset -> find data in list

THANK YOU!

감사합니다!