# DOCUMENTATION

## COMPUTER ORGANISATION

### END SEM ASSIGNMENT (BONUS TASK)

### TWO LEVEL CACHE IMPLEMENTATION

**Name : ABHISHEK CHATURVEDI**
**Roll no. : 2019401**
**Group no. : 3**

# ASSIGNMENT OVERVIEW

- **Implementation of two level cache for direct, fully associative and n way set associative mapping which allows searching and loading data into cache memory.**

- **Programming language: Python 3**

- **The algorithm used for replacement is LRU (Least Recently Used).**

- **The algorithm used for cache inclusion policy is NINE(Neither inclusive nor exclusive)**

- **Inputs:**
    - **Main memory size**
    - **No. of cache lines**
    - **Block size**
    - **'n' (only for n way set associative mapping)**
    - **Read(r) or write(w) operation**
    - **Address (if read)/ address and data(if write)**

- **Outputs:**
    - **Both level one and level two cache after each operation.**
    - **Write operation:**
        - **Address breakdown depending on the type of mapping (block no., set no., line no., tag, word no.)**
        - **Address found/not found in cache**
        - **If the address is found then its location in cache(line no., set no. etc.)**
        - **If the address is not found then location in cache memory where data is to be loaded(line no., set no. etc.) and the block no. which is to be loaded**
        - **Message that the word no. at the given address is updated with input data**
    - **Read operation:**
        - **Address breakdown depending on the type of mapping ( i.e. block no., set no., line no., tag, word no.)**
        - **Cache hit/miss i.e. address found/not found**
        - **Data/value at the address**
        - **Location in cache memory where data was found depending on the type of mapping( i.e. line no., set no. etc.)**

## ASSUMPTIONS

- The main memory size and block size are in terms of number of words they can store.
- Number of cache lines in input is for level one cache.
- The block size of first and second level cache are assumed to be equal.
- The type of mapping for the level one and level two cache is assumed to be the same.
- The value of ‘n’ in n way set associative mapping is assumed to be equal for both the levels of cache.
- Word size length is assumed to be 32 bits (1Word=4Bytes) but the program will run successfully for even 16 bits or 64 bits and more.
- The main memory size, block size, number of cache lines and ‘n’ are all assumed in powers of 2, i.e. the minimum value of these variables can be 2.
- SIze of level two cache is twice the size of level one cache.
- Initially all the values in tag and cache lines are initialised with “NULL” string.

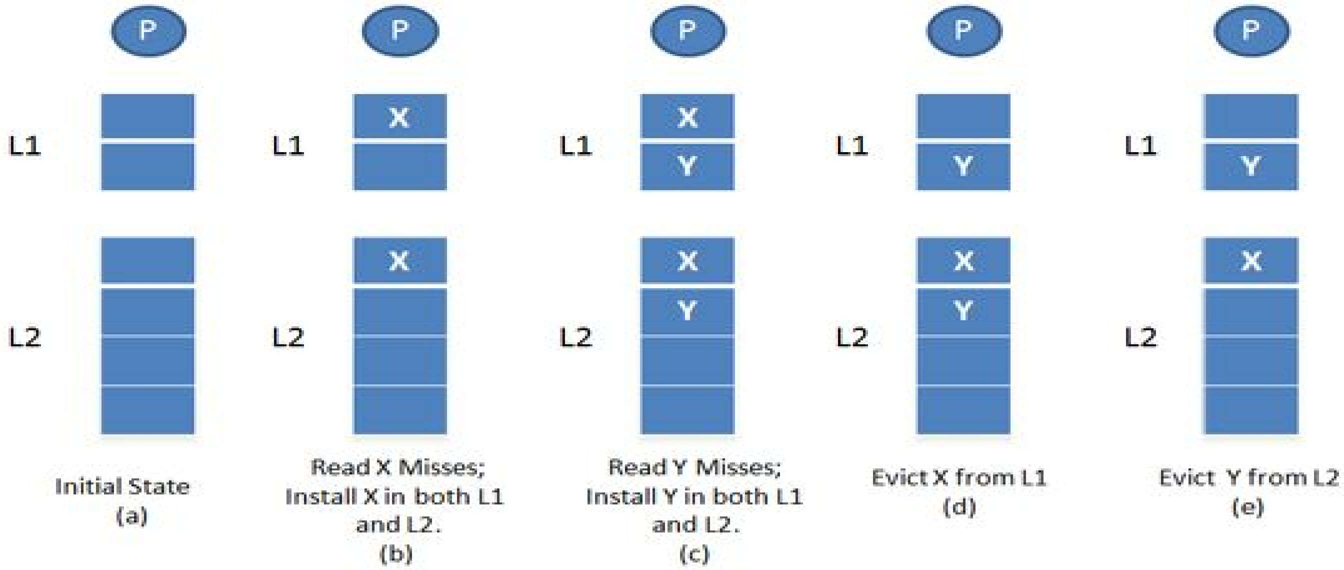
## ERROR HANDLING

- If the length of address is not equal to the number of bits required to express the main memory size in power of 2 then the program gives output “INVALID ADDRESS” and asks for input again.
- The size of either of the caches cannot be equal to or greater than the main memory size, if this is not the case then the program gives inbuilt python error.

## NINE INCLUSION POLICY

- In this type of policy if there is a write operation then the block is searched for in level one and level two cache and if the block is found in both the levels then the given data is updated at the address in both the cache levels.
- If the block is present in level one cache only then data at the given address is written in level one cache only and the block is not loaded in level two cache.
- If the block is not found in level one cache but it is present in level two cache then that block is loaded in level one cache from level two cache before the data is written at the address in both the cache levels.
- If the block is not found in any level of cache then the block is placed in both cache levels and then the data is updated at the address.

- In this policy there is no connection between the two levels of cache at the time of eviction of a block i.e, the removal of a block in one cache level is independent of the other.
- If a block is to be evicted from second level then there is no back invalidation like in case of inclusive policy i.e, a block is removed from level two cache only even if that block is present in the first level cache.

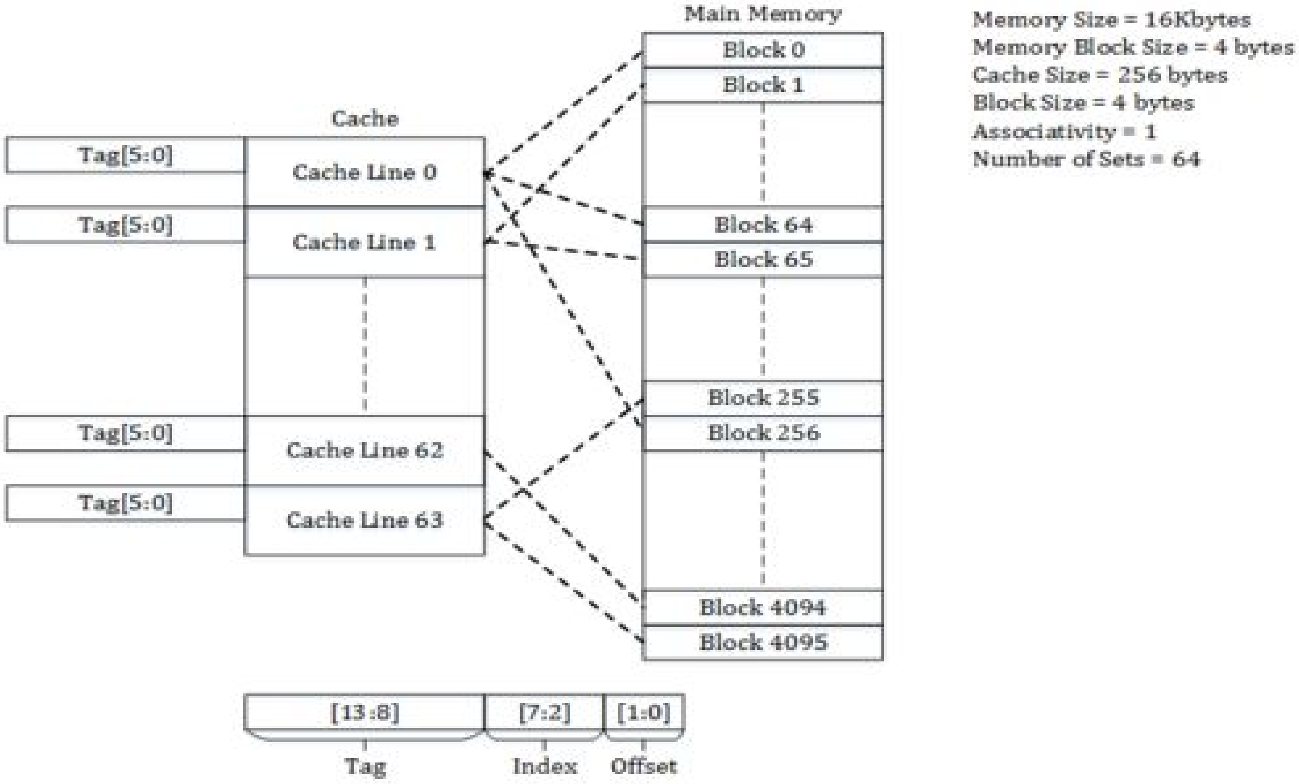


# DIRECT MAPPING

## CONCEPT:

- In this type of mapping each block of main memory maps into only one specific cache line in the cache memory.
- The physical address is split up into three parts i.e, the block offset(same for both cache), the line no.(different for two levels of cache) and the tag(different for two levels of cache).
- The block offset is the index of the word in the block.
- The line no. bits and tag bits make up the block no. bits and this block no. is used to determine the line no. where this block will be placed in cache memory.
- The line no. is equal to (block no.(in decimal)) modulo (no. of cache lines) i.e, the remainder when block no. is divided by the no. of cache lines.
- Firstly, the address' tag bits are compared to the tag bits at the cache line number in level one cache.
- If the two tags match then it is a hit and data is returned.

- If tag bits don’t match in level one tags then level two tag bits are compared to address tag bits and if the tag bits match it is a hit otherwise miss.
- If it is a hit in level two cache then the block is loaded in level one cache also before the data at the address is returned.
- If it is a miss then the block is brought from main memory and loaded in both level one and level two cache.
- If it is a miss and the cache line is already occupied by another block then that block is evicted out and the current block is placed in the cache line.
- If it is a miss and the cache line is empty then the block is simply added to the cache line.



## ADVANTAGES

- This mapping is faster since tags are compared at a particular line no. in cache and there is no need to search through the whole cache.
- The replacement policy is simple and easy to implement.
- Cheap hardware implementation.

## DISADVANTAGES

- Low cache hit rate.

## WORKING OF THE PROGRAM

- **The program divides the physical address into tag, line no. and block offset.**
- **Firstly, the program simply compares the tag of the given address and the tag of the line no. where block is to be placed in level one cache.**
- **If the tag bits match and it is a write operation then data is written at the address and then the tag bits of line no. in level two cache is compared if the block is present then value is also written at the address in level two cache.**
- **If the tag bits match in level one cache then data at the address is returned.**
- **If the tag bits do not match in level one cache then level two cache's line no. tag bits are compared and if the tag bits match and it is write operation then the block is loaded in level one cache also and data at the address is updated in both the levels of cache.**
- **If tag bits match and it is read operation data at the block offset in the block is returned.**
- **If the tag bits don't match even in level two cache and it is a write operation then the block is loaded in both levels of cache and then data at the address is written.**

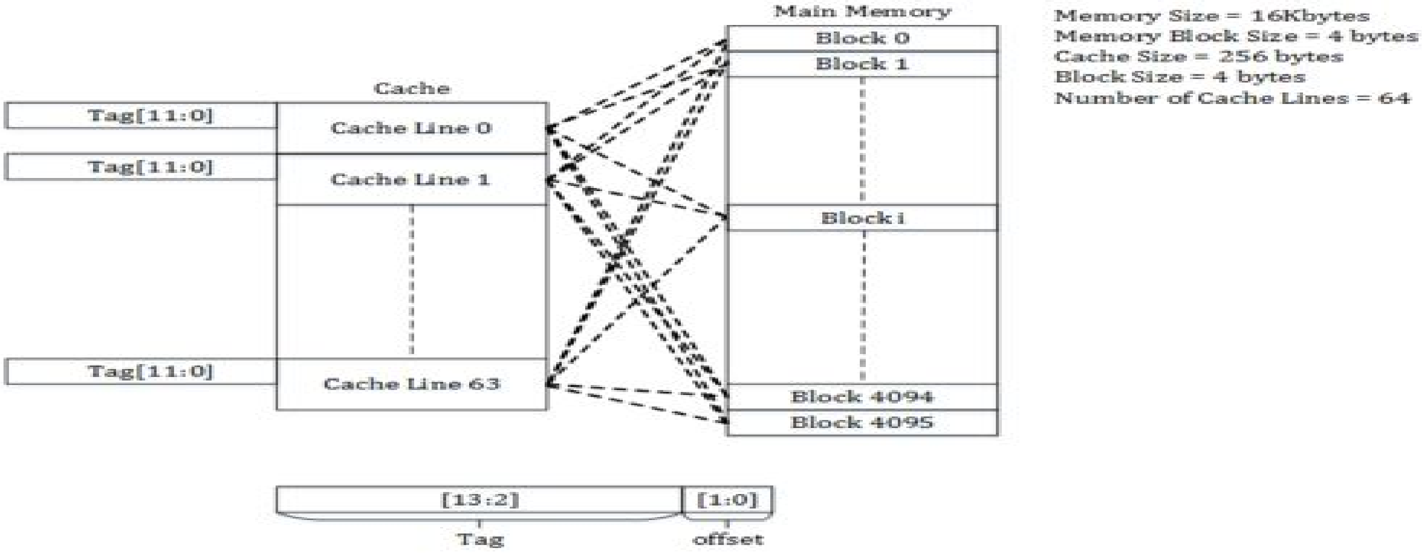
## ANALYSIS OF THE CODE

- **Two arrays(list in python) are used as data structures for storing tag values and blocks separately.**
- **Since the program only compares the tags at a given line no. it takes constant time i.e, O(1).**
- **The program also removes and places a new block and writes an address which is done in constant time O(1).**
- **Therefore, the overall time complexity of the program is O(1).**
- **The space complexity of the program is O(n) since tag array and cache memory are arrays of size n. (n=no. of cache lines in level two cache)**

# FULLY ASSOCIATIVE MAPPING

## CONCEPT:

- **In this type of mapping a block from main memory can be mapped to any line in cache memory given that the line is free.**
- **The physical address is split into only two parts i.e, the tag or block no. and block offset.**
- **The block offset is the index of the word in the block.**
- **The tag of the given address is compared to the tag of every cache line in level one cache one by one until the tags match or all the cache line's tag is compared.**
- **If there is a match of tags then it is a hit and the data at the block offset is returned.**
- **If the tags do not match then the address tag is compared to the tag of every line in level two cache.**
- **If the tags do not match in even level two cache then the block is placed in both level one and level two caches and then the data at the block offset in the block is returned.**
- **If the tags match in level two cache then the block is loaded in level one cache also before the data at the address is returned.**
- **If there is no matching of tags and there is an empty cache line then the block is simply added to cache memory.**
- **If there is no matching of tags and all the cache lines are occupied then a block is evicted from the cache memory and a new block is placed in the cache memory.**
- **The block which is to be evicted is decided on the basis of replacement policy (LRU in this case).**



## ADVANTAGES

- **Better hit rate.**
- **Since there is full flexibility of storing a block in any cache line it ensures complete utilisation of cache.**
- **A wide variety of replacement algorithms can be used for replacement.**

## DISADVANTAGES

- **Since we need to compare the tag of each cache line the search and placement policy is slow.**
- **Expensive hardware implementation since a lot of comparisons are involved.**

## WORKING OF THE CODE

- **The program firstly divides the physical address into two parts i.e, block offset and tag/block no.**
- **Then the program iterates over all the cache lines in the level one cache and compares the tag bits.**
- **If the tag bits of address and tag bits of a cache line match it is a hit and the data at the block offset in the block is returned in read operation and in write the given data is written at the block offset in the block.**
- **If the tag bits match in level one cache and it is write operation then the tag bits in level two cache are compared to address tag bits and if the tag bits match then data is updated at the address in level two cache also.**
- **If the tag bits do not match in level one cache then the tag bits in level two cache are compared.**
- **If tag bits in level two cache match then the block is loaded in level one cache and then data is written at the address in both the levels of cache if it is write operation.**
- **If the tag bits do not match even in level two cache and it is a write input then the block is uploaded in both level one and level two cache and then data is updated at the address.**
- **If it is a miss and it is write operation and all the cache lines are filled then a block is evicted from the cache memory based on LRU replacement algorithm and the new block is placed in the cache memory and the given data is written at block offset of the block.**

## ANALYSIS OF CODE

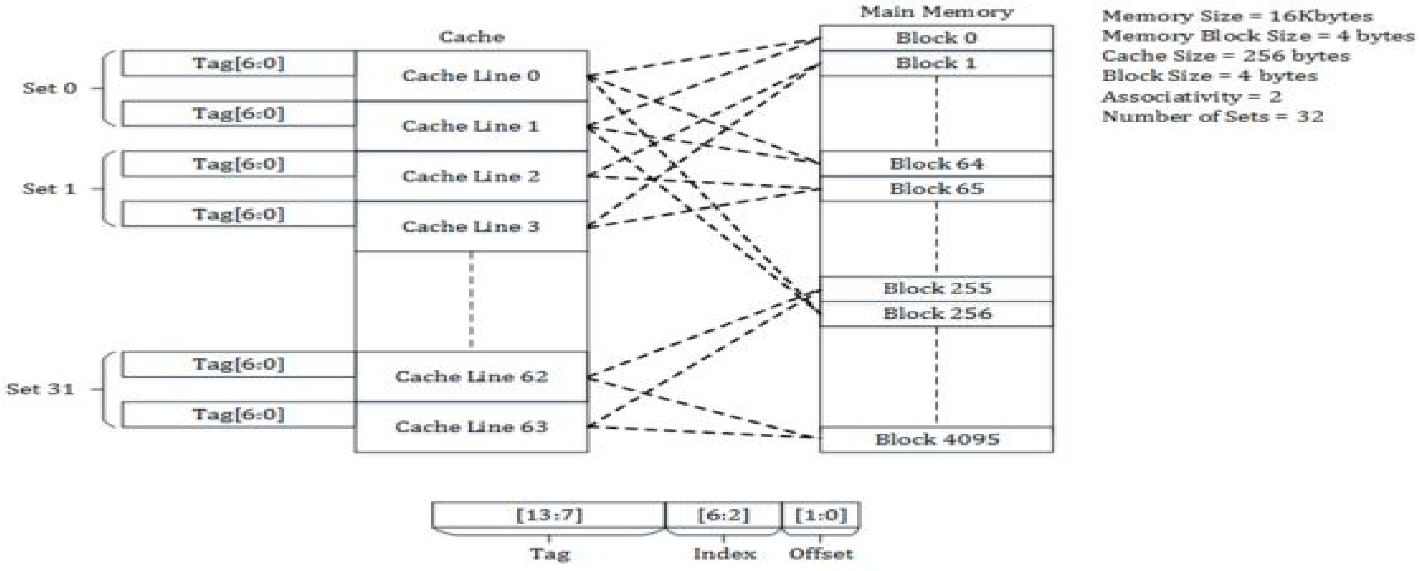
- **The data structures used are a dictionary(hashmap in other common languages) and a queue implemented using a doubly linked list.**
- **The queue is used for maintaining the order for replacement in LRU algorithm. It has most recently used blocks in the front and least recently used block in the rear.**

- Since the program looks up for a block no. in hashmap/dictionary it is done in constant time of O(1).
- Also deleting/removing and adding/replacing a block in cache memory takes constant time O(1).
- Therefore, the overall time complexity of the program is O(1).
- The space complexity of the program is O(n) because there can be n blocks in the cache memory. (n=no of cache lines in level two cache)

# N WAY SET ASSOCIATIVE MAPPING

## CONCEPT

- This type of mapping is a hybrid of fully associative mapping and direct mapping.
- In this type of mapping the level one and level two caches are divided into sets each containing n cache lines.
- Therefore, the number of sets in each cache level is equal to quotient when (number of cache lines in that cache) is divided by (n).
- A block from the main memory can map in a specific set only but at any line in that set given that the line is free.
- The physical address is divided into three parts namely tag, set number and block offset.
- The block offset is the index of the word in the block.
- The tag bits of the address are compared to tag bits to each cache line in the set (derived from set bits in the address) until there is a match of tag bits or every cache line's tag is compared.
- If the tag bits match it is a hit otherwise miss.



## ADVANTAGES

- **There is a wide variety of replacement policies which could be used.**

## DISADVANTAGES

- **It does not utilise all the available cache lines so the cache misses are more.**

## WORKING OF THE CODE

- **The program firstly divides the physical address into three parts i.e, block offset, set number(different for the two cache levels) and tag(different for two levels of cache).**
- **Then the program iterates over all the cache lines in the set(derived from the address) in the level one cache and compares the tag bits.**
- **If the tag bits of address and tag bits of a cache line match in level one cache it is a hit and the data at the block offset in the block is returned in read operation and in write the given data is written at the block offset in the block.**
- **If tag bits do not match in level one cache then the tag bits of cache in lines of the set of level two cache are compared.**
- **If the tag bits match and it is a write operation then the block is loaded into level one cache and then the given data is written at the address in both level one and level two cache.**
- **If the tag bits do not match then it is a miss and in write operation the block is placed in both first and second level cache and then data is updated at the given address in both the cache.**
- **If tag bits do not match and it is a write operation and all the cache lines in the set are filled then a block is evicted from the set based on LRU replacement algorithm and the new block is placed in the cache memory and the given data is written at block offset of the block.**
- **If it is a miss and it is a write operation and all the cache lines in the set are not filled then the block is simply added to the set and the given data is written at block offset of the block.**

## ANALYSIS OF CODE

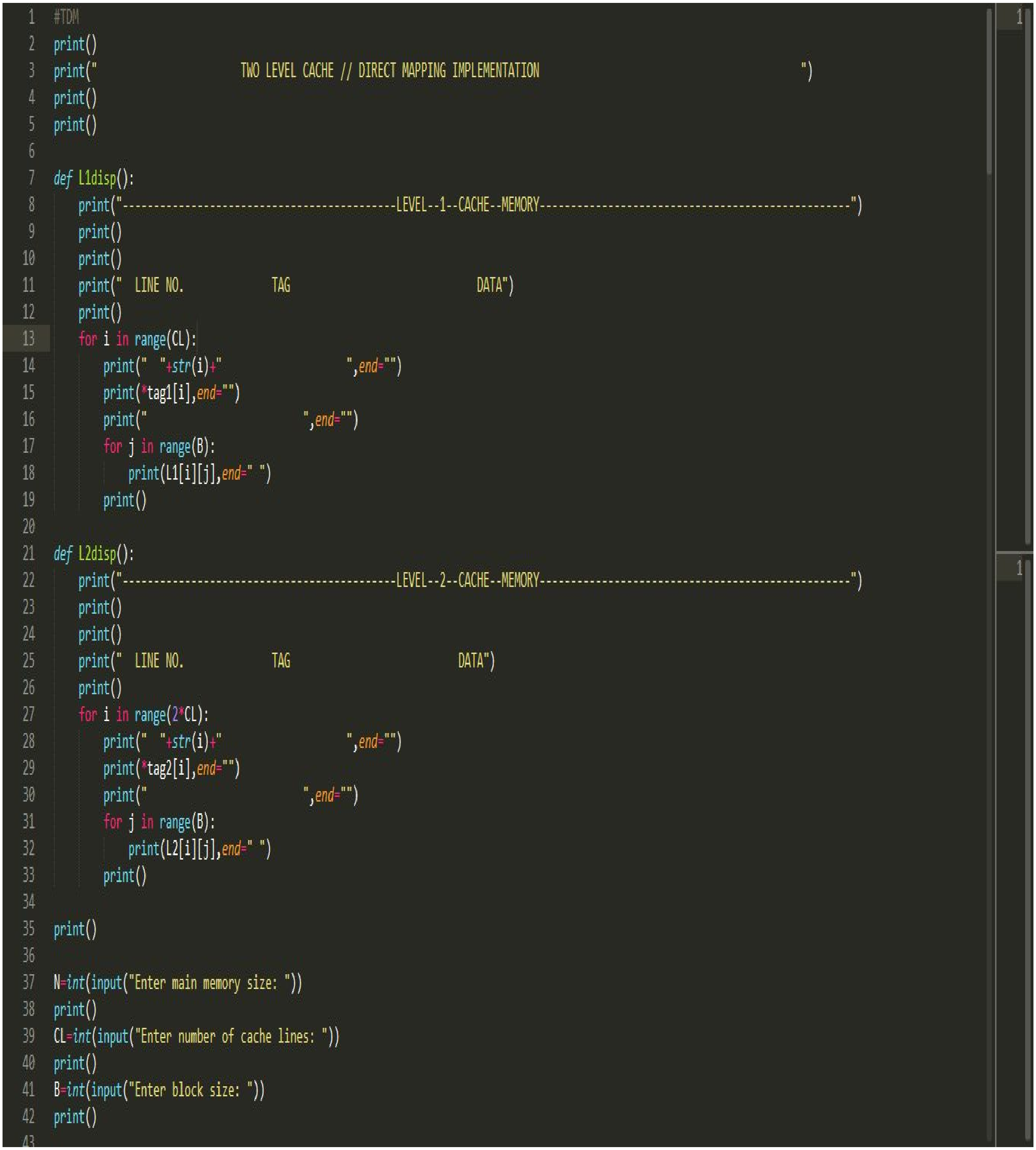
- **Four arrays(list in python) are used as data structures for storing tag values and blocks separately for level one and level two cache.**
- **Since the program only compares all the cache line’s tags in a given set of both the cache levels it takes linear*linear=square time i.e, O(n^2). (n=no. of cache lines in the set)**
- **The program also removes and places a new block and writes an address which is done in constant time O(1).**
- **Therefore, the overall time complexity of the program is O(n^2).**

- **The space complexity of the program is O(n) since tag array and cache memory are arrays of size n. (n=no. of cache lines)**

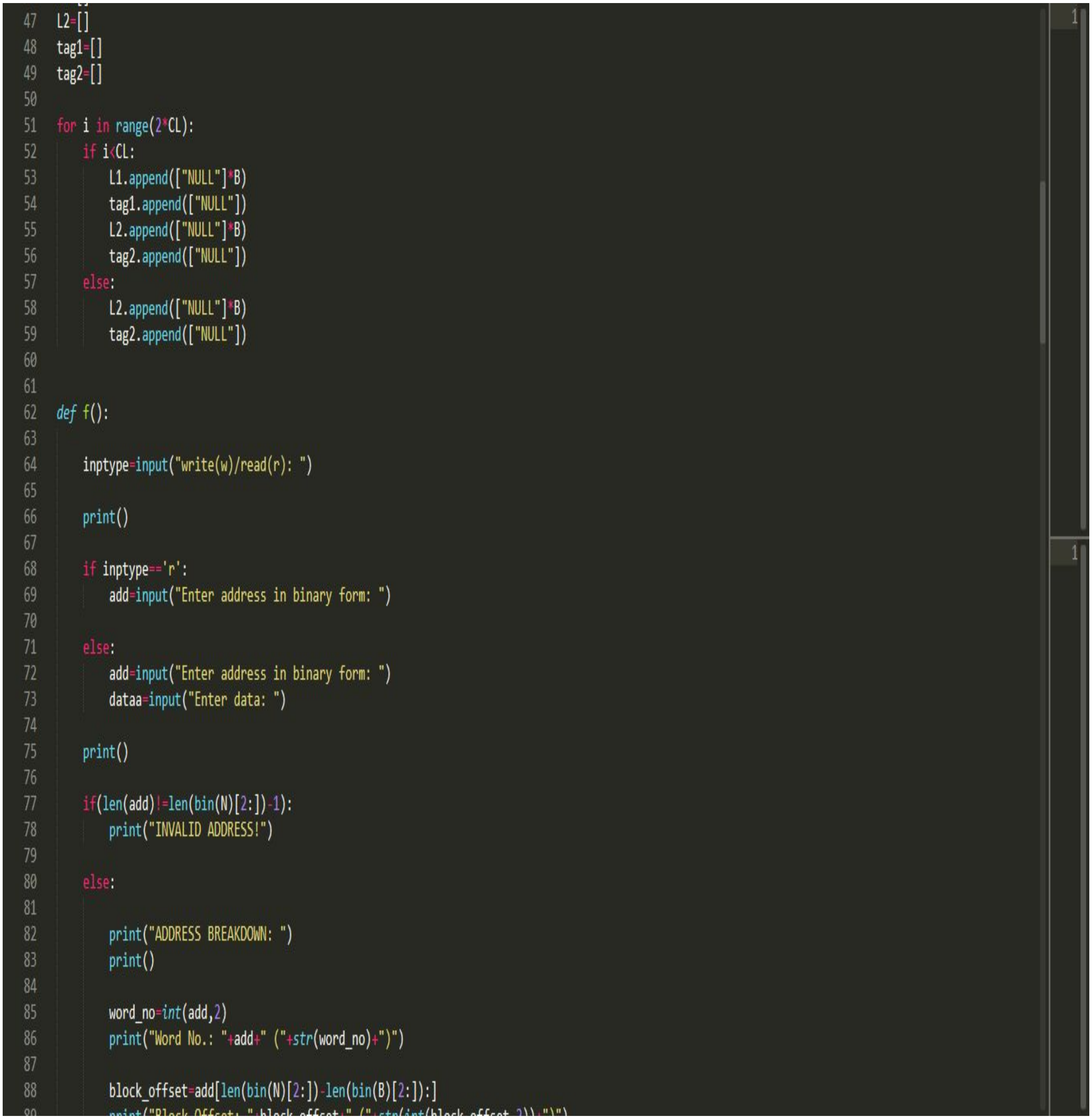
## REFERENCES:

- https://en.wikipedia.org/wiki/Cache_placement_policies
- https://en.wikipedia.org/wiki/Cache_inclusion_policy#NINE_Policy
- https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

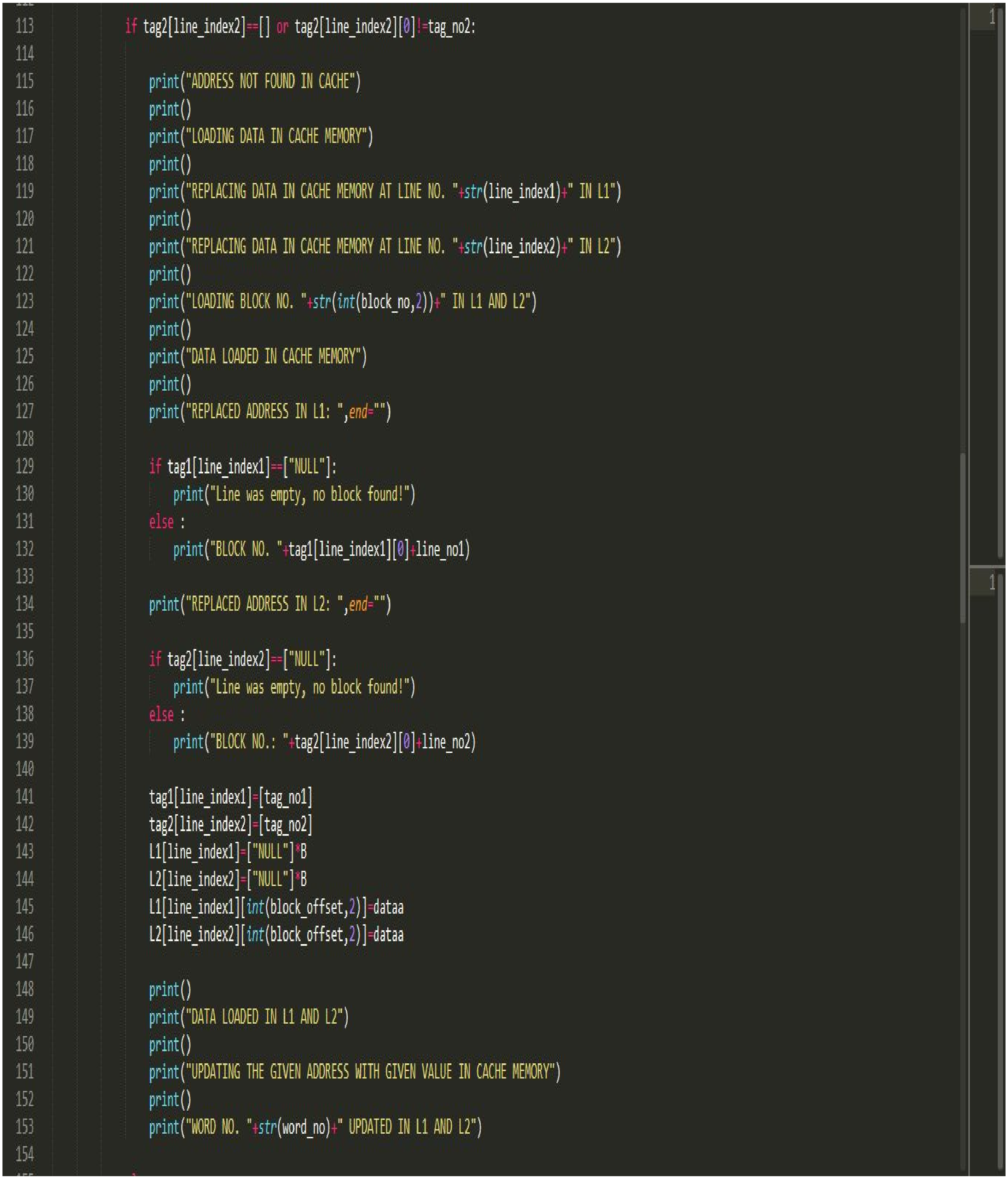
*******************(screenshots of the codes is attached below)*******************



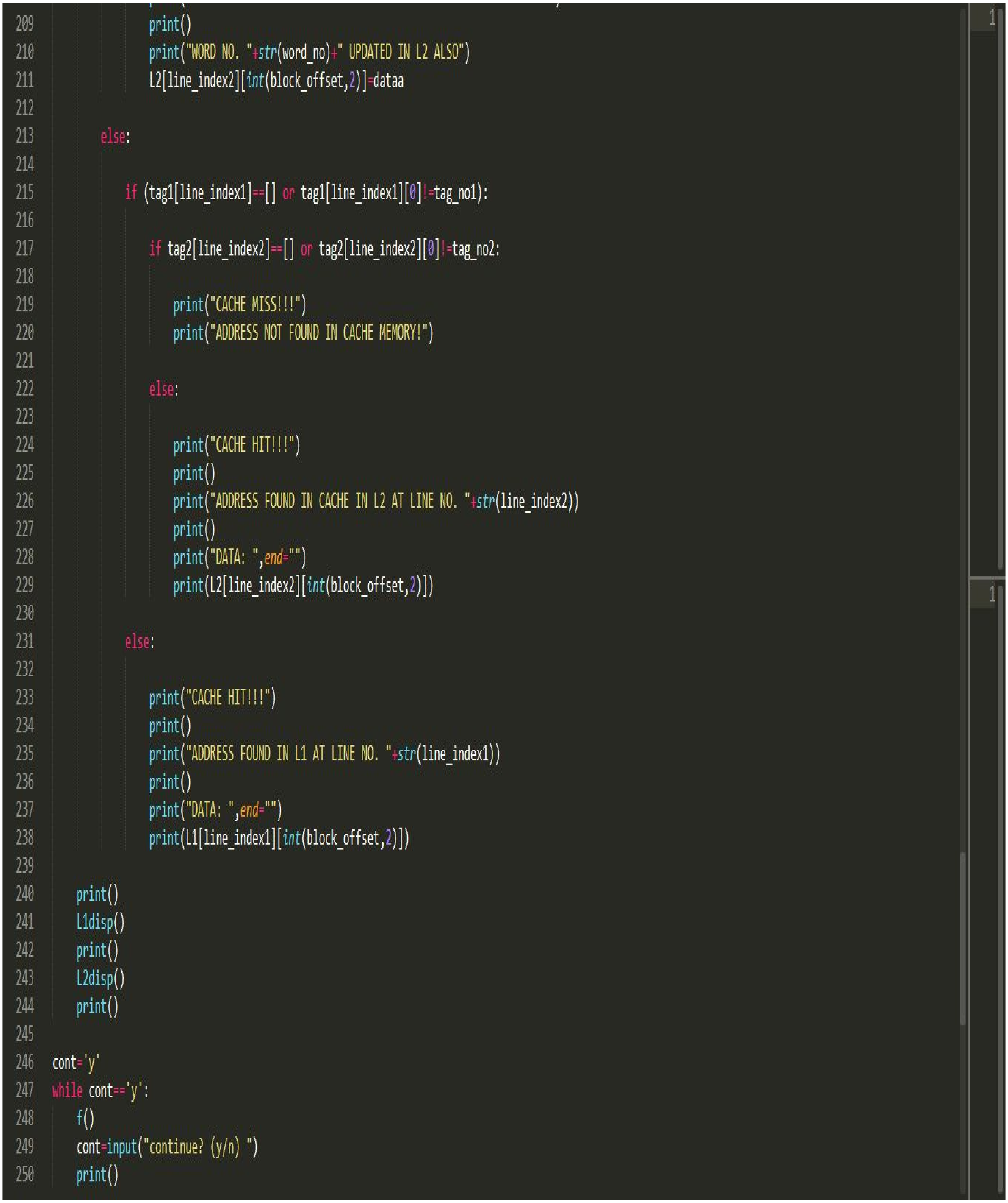
```
#TDM
print()
print("                       TWO LEVEL CACHE // DIRECT MAPPING IMPLEMENTATION                                           ")
print()
print()

def L1disp():
    print("---------------------------------------------LEVEL--1--CACHE--MEMORY---------------------------------------------------")
    print()
    print()
    print("  LINE NO.              TAG                              DATA")
    print()
    for i in range(CL):
        print("  "+str(i)+"                    ",end="")
        print(*tag1[i],end="")
        print("                         ",end="")
        for j in range(B):
            print(L1[i][j],end=" ")
        print()

def L2disp():
    print("---------------------------------------------LEVEL--2--CACHE--MEMORY---------------------------------------------------")
    print()
    print()
    print("  LINE NO.              TAG                           DATA")
    print()
    for i in range(2*CL):
        print("  "+str(i)+"                    ",end="")
        print(*tag2[i],end="")
        print("                         ",end="")
        for j in range(B):
            print(L2[i][j],end=" ")
        print()

print()

N=int(input("Enter main memory size: "))
print()
CL=int(input("Enter number of cache lines: "))
print()
B=int(input("Enter block size: "))
print()
```



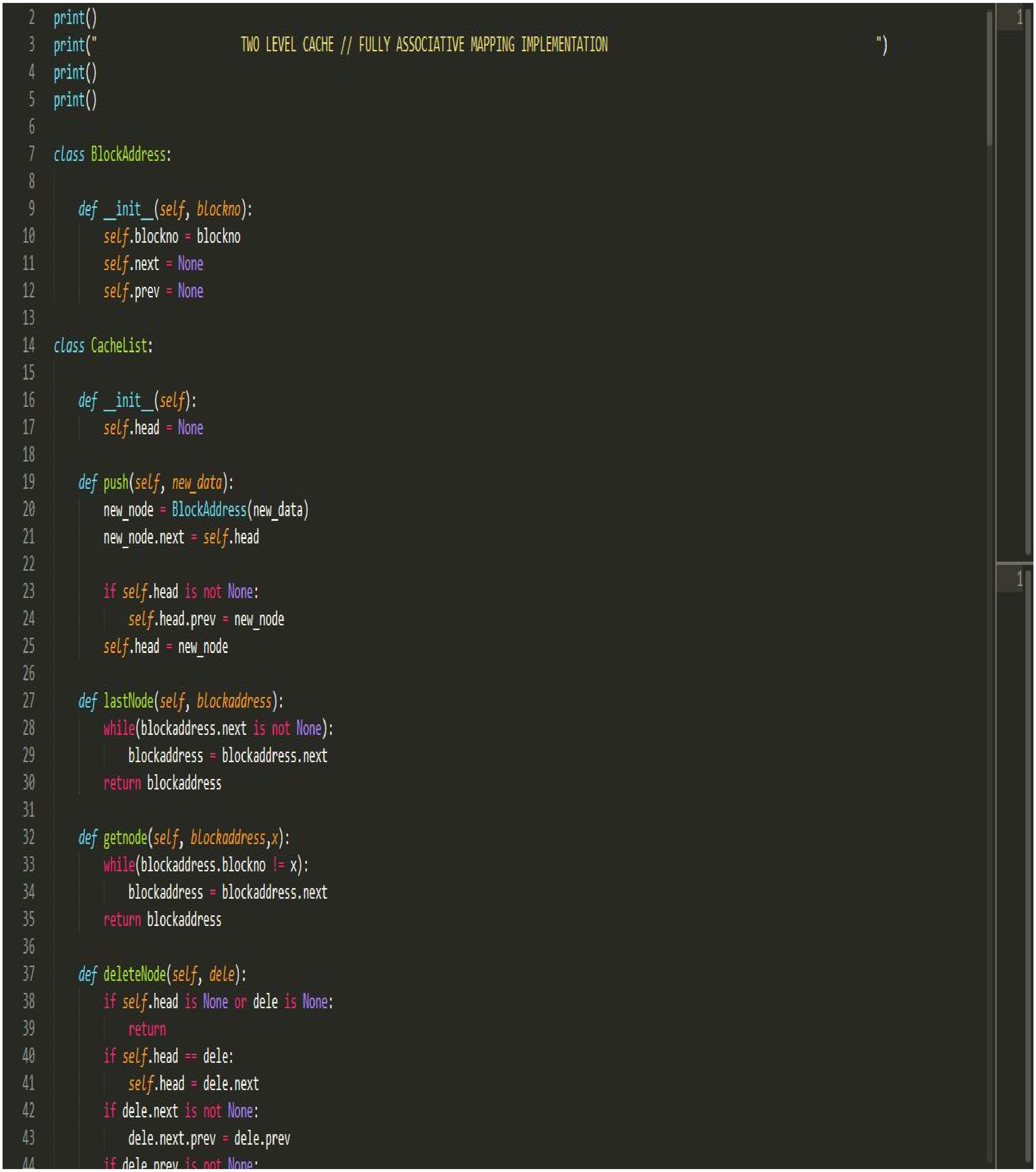
```python
L2=[]
tag1=[]
tag2=[]

for i in range(2*CL):
    if i<CL:
        L1.append(["NULL"]*B)
        tag1.append(["NULL"])
        L2.append(["NULL"]*B)
        tag2.append(["NULL"])
    else:
        L2.append(["NULL"]*B)
        tag2.append(["NULL"])


def f():

    inptype=input("write(w)/read(r): ")

    print()

    if inptype=='r':
        add=input("Enter address in binary form: ")

    else:
        add=input("Enter address in binary form: ")
        dataa=input("Enter data: ")

    print()

    if(len(add)!=len(bin(N)[2:])-1):
        print("INVALID ADDRESS!")

    else:

        print("ADDRESS BREAKDOWN: ")
        print()

        word_no=int(add,2)
        print("Word No.: "+add+" ("+str(word_no)+")")

        block_offset=add[len(bin(N)[2:])-len(bin(B)[2:]):]
        print("Block Offset: "+block_offset+" ("+str(int(block_offset,2))+")")
```



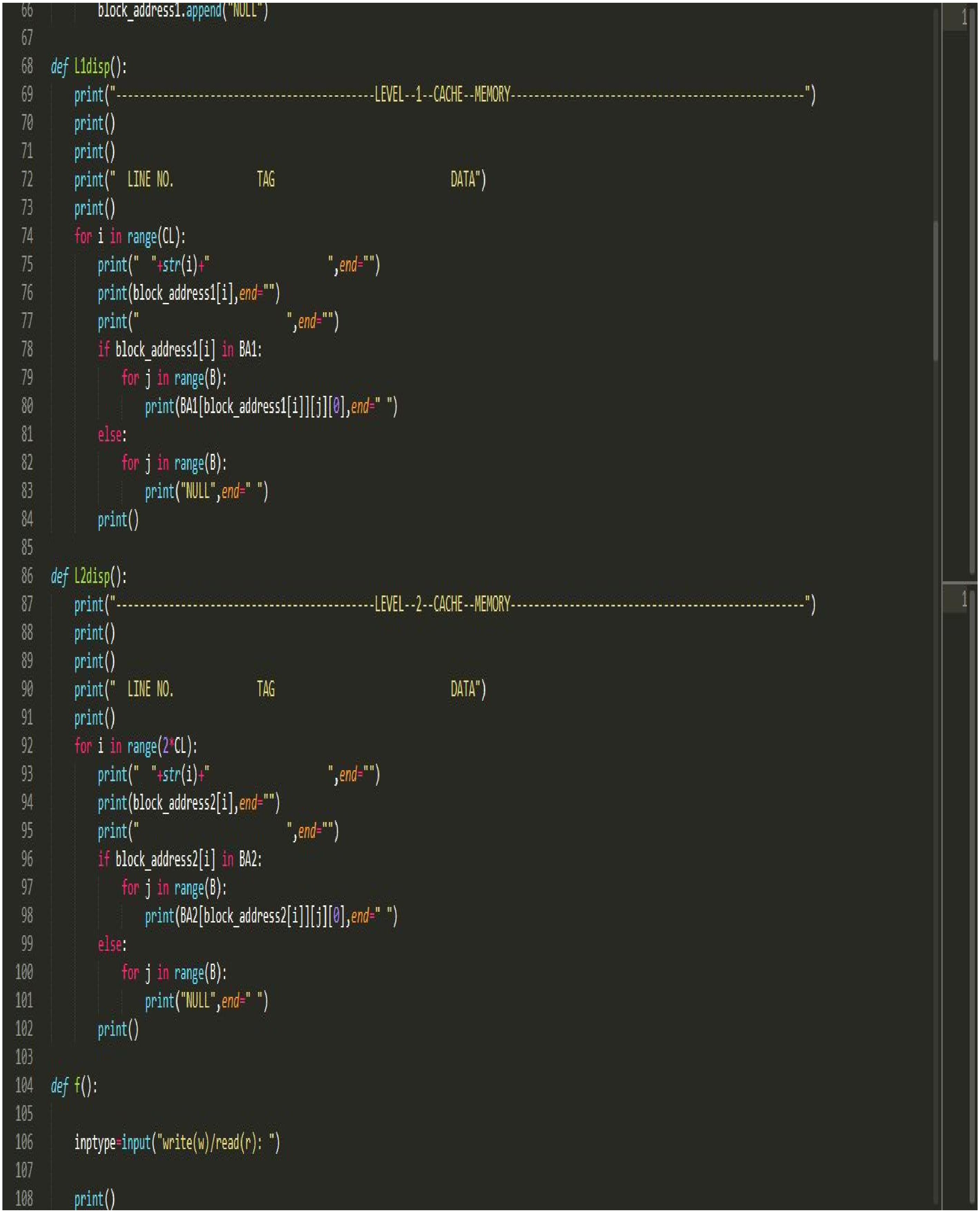
```
            if tag2[line_index2]==[] or tag2[line_index2][0]!=tag_no2:

                print("ADDRESS NOT FOUND IN CACHE")
                print()
                print("LOADING DATA IN CACHE MEMORY")
                print()
                print("REPLACING DATA IN CACHE MEMORY AT LINE NO. "+str(line_index1)+" IN L1")
                print()
                print("REPLACING DATA IN CACHE MEMORY AT LINE NO. "+str(line_index2)+" IN L2")
                print()
                print("LOADING BLOCK NO. "+str(int(block_no,2))+" IN L1 AND L2")
                print()
                print("DATA LOADED IN CACHE MEMORY")
                print()
                print("REPLACED ADDRESS IN L1: ",end="")

                if tag1[line_index1]==["NULL"]:
                    print("Line was empty, no block found!")
                else :
                    print("BLOCK NO. "+tag1[line_index1][0]+line_no1)

                print("REPLACED ADDRESS IN L2: ",end="")

                if tag2[line_index2]==["NULL"]:
                    print("Line was empty, no block found!")
                else :
                    print("BLOCK NO.: "+tag2[line_index2][0]+line_no2)

                tag1[line_index1]=[tag_no1]
                tag2[line_index2]=[tag_no2]
                L1[line_index1]=["NULL"]*B
                L2[line_index2]=["NULL"]*B
                L1[line_index1][int(block_offset,2)]=dataa
                L2[line_index2][int(block_offset,2)]=dataa

                print()
                print("DATA LOADED IN L1 AND L2")
                print()
                print("UPDATING THE GIVEN ADDRESS WITH GIVEN VALUE IN CACHE MEMORY")
                print()
                print("WORD NO. "+str(word_no)+" UPDATED IN L1 AND L2")

```



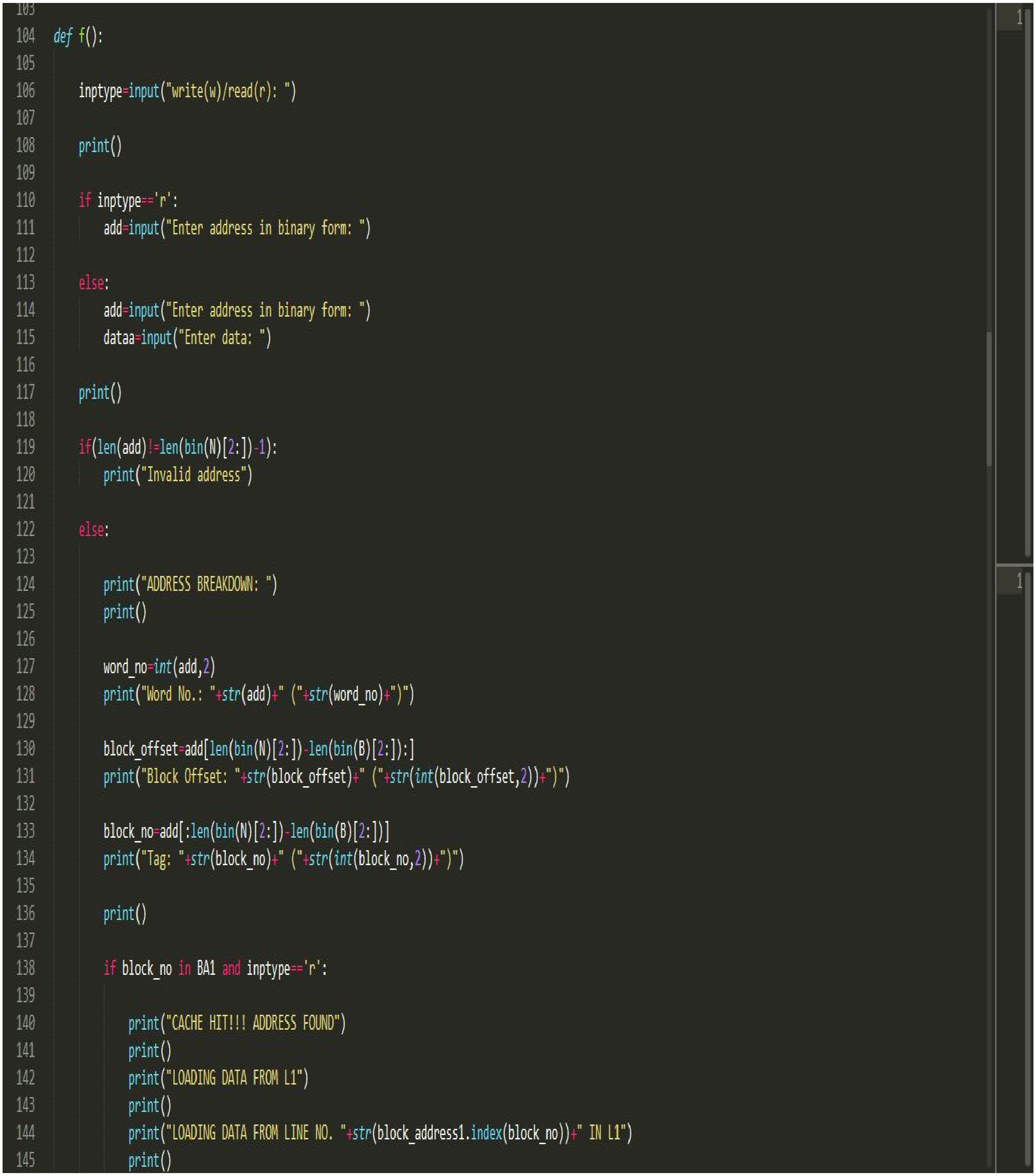
```
                print()
                print("WORD NO. "+str(word_no)+" UPDATED IN L2 ALSO")
                L2[line_index2][int(block_offset,2)]=dataa

        else:

            if (tag1[line_index1]==[] or tag1[line_index1][0]!=tag_no1):

                if tag2[line_index2]==[] or tag2[line_index2][0]!=tag_no2:

                    print("CACHE MISS!!!")
                    print("ADDRESS NOT FOUND IN CACHE MEMORY!")

                else:

                    print("CACHE HIT!!!")
                    print()
                    print("ADDRESS FOUND IN CACHE IN L2 AT LINE NO. "+str(line_index2))
                    print()
                    print("DATA: ",end="")
                    print(L2[line_index2][int(block_offset,2)])

            else:

                print("CACHE HIT!!!")
                print()
                print("ADDRESS FOUND IN L1 AT LINE NO. "+str(line_index1))
                print()
                print("DATA: ",end="")
                print(L1[line_index1][int(block_offset,2)])

    print()
    L1disp()
    print()
    L2disp()
    print()

cont='y'
while cont=='y':
    f()
    cont=input("continue? (y/n) ")
    print()
```



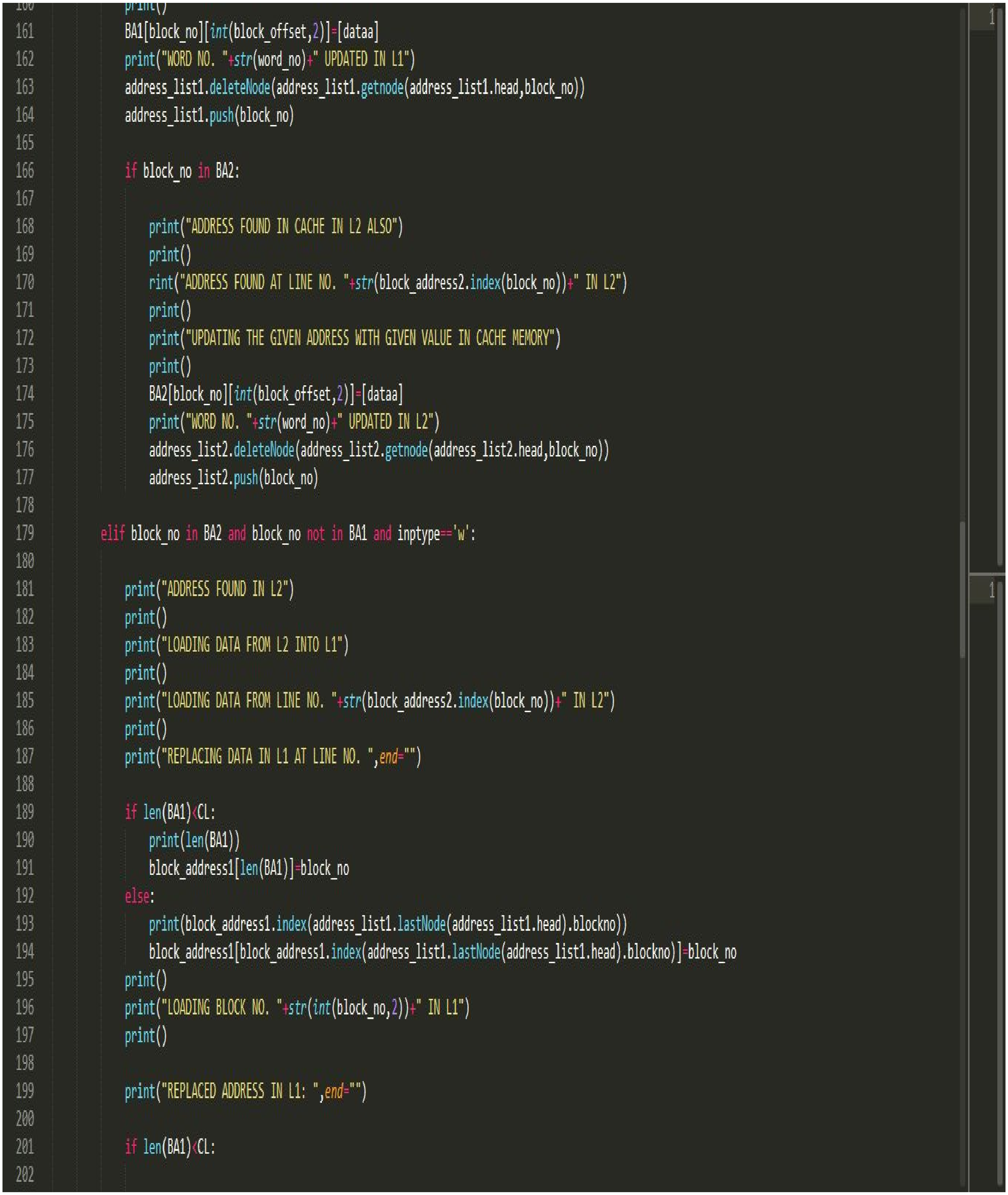
```
print()
print("                      TWO LEVEL CACHE // FULLY ASSOCIATIVE MAPPING IMPLEMENTATION                                           ")
print()
print()

class BlockAddress:

    def __init__(self, blockno):
        self.blockno = blockno
        self.next = None
        self.prev = None

class CacheList:

    def __init__(self):
        self.head = None

    def push(self, new_data):
        new_node = BlockAddress(new_data)
        new_node.next = self.head

        if self.head is not None:
            self.head.prev = new_node
        self.head = new_node

    def lastNode(self, blockaddress):
        while(blockaddress.next is not None):
            blockaddress = blockaddress.next
        return blockaddress

    def getnode(self, blockaddress,x):
        while(blockaddress.blockno != x):
            blockaddress = blockaddress.next
        return blockaddress

    def deleteNode(self, dele):
        if self.head is None or dele is None:
            return
        if self.head == dele:
            self.head = dele.next
        if dele.next is not None:
            dele.next.prev = dele.prev
        if dele.prev is not None:
```



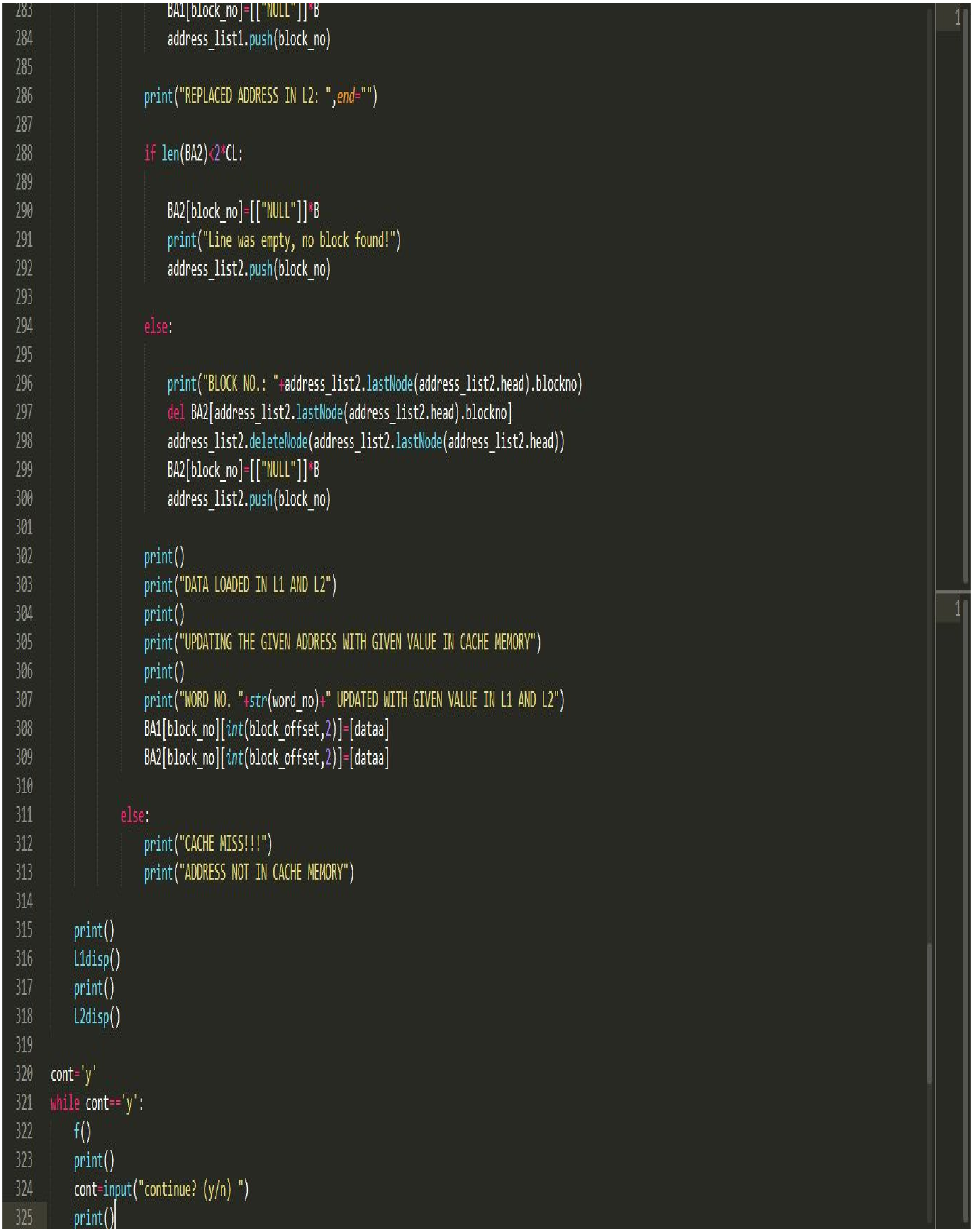
```
        block_address1.append("NULL")

def L1disp():
    print("--------------------------------------------LEVEL--1--CACHE--MEMORY--------------------------------------------------")
    print()
    print()
    print("  LINE NO.            TAG                            DATA")
    print()
    for i in range(CL):
        print("  "+str(i)+"                    ",end="")
        print(block_address1[i],end="")
        print("                        ",end="")
        if block_address1[i] in BA1:
            for j in range(B):
                print(BA1[block_address1[i]][j][0],end=" ")
        else:
            for j in range(B):
                print("NULL",end=" ")
        print()

def L2disp():
    print("--------------------------------------------LEVEL--2--CACHE--MEMORY--------------------------------------------------")
    print()
    print()
    print("  LINE NO.            TAG                            DATA")
    print()
    for i in range(2*CL):
        print("  "+str(i)+"                    ",end="")
        print(block_address2[i],end="")
        print("                        ",end="")
        if block_address2[i] in BA2:
            for j in range(B):
                print(BA2[block_address2[i]][j][0],end=" ")
        else:
            for j in range(B):
                print("NULL",end=" ")
        print()

def f():

    inptype=input("write(w)/read(r): ")

    print()
```



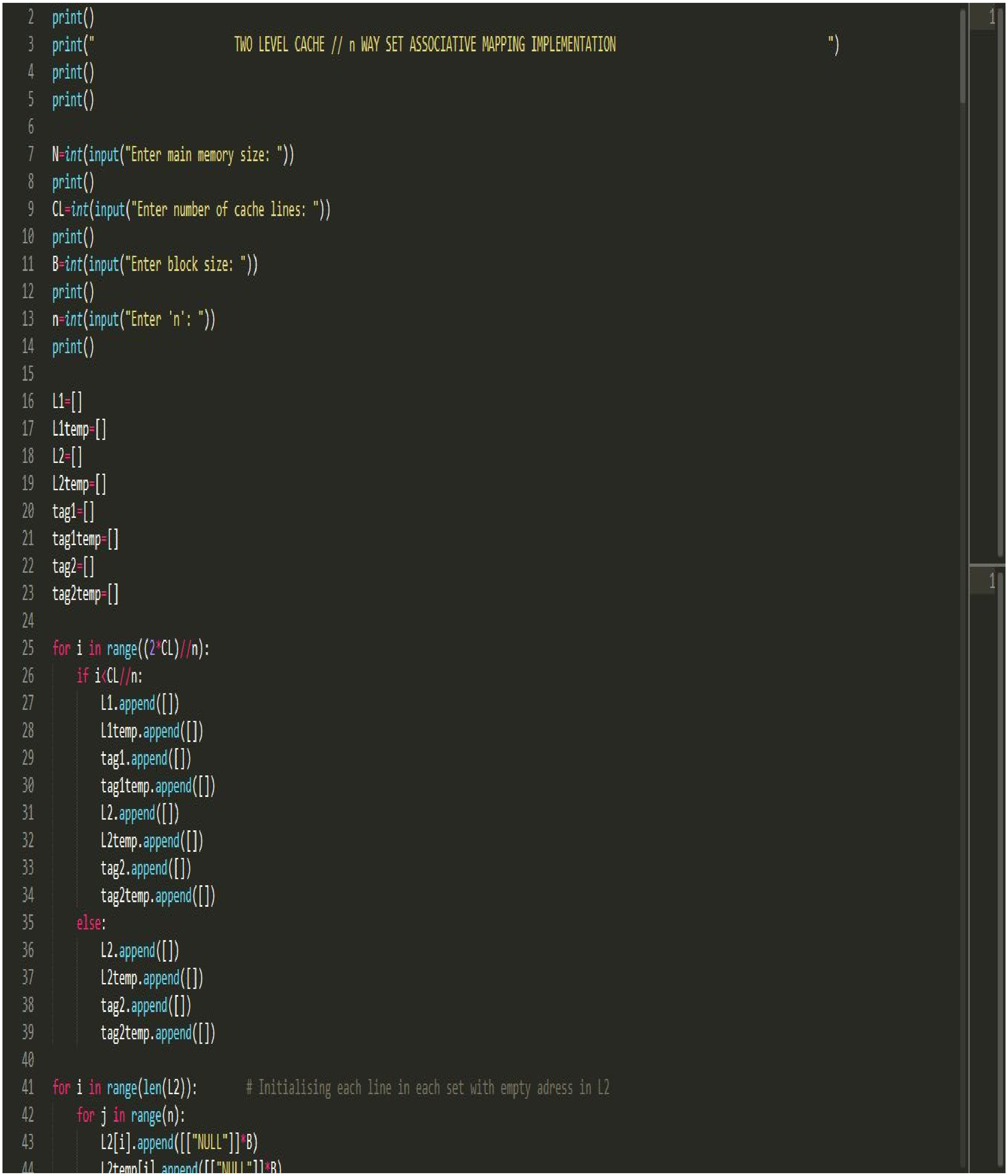
```
def f():

    inptype=input("write(w)/read(r): ")

    print()

    if inptype=='r':
        add=input("Enter address in binary form: ")

    else:
        add=input("Enter address in binary form: ")
        dataa=input("Enter data: ")

    print()

    if(len(add)!=len(bin(N)[2:])-1):
        print("Invalid address")

    else:

        print("ADDRESS BREAKDOWN: ")
        print()

        word_no=int(add,2)
        print("Word No.: "+str(add)+" ("+str(word_no)+")")

        block_offset=add[len(bin(N)[2:])-len(bin(B)[2:]):]
        print("Block Offset: "+str(block_offset)+" ("+str(int(block_offset,2))+")")

        block_no=add[:len(bin(N)[2:])-len(bin(B)[2:])]
        print("Tag: "+str(block_no)+" ("+str(int(block_no,2))+")")

        print()

        if block_no in BA1 and inptype=='r':

            print("CACHE HIT!!! ADDRESS FOUND")
            print()
            print("LOADING DATA FROM L1")
            print()
            print("LOADING DATA FROM LINE NO. "+str(block_address1.index(block_no))+" IN L1")
            print()
```



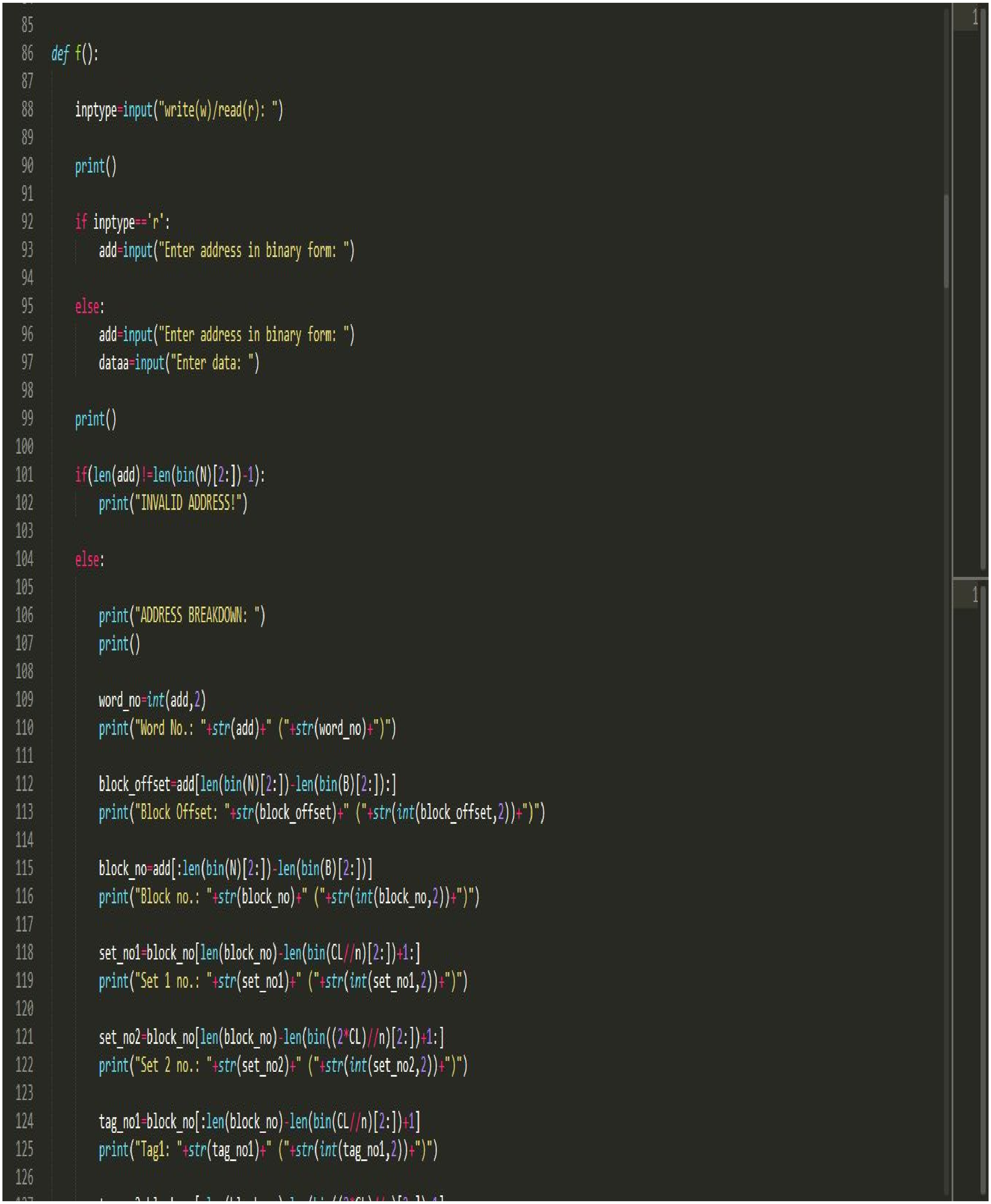
```
            print()
            BA1[block_no][int(block_offset,2)]=[dataa]
            print("WORD NO. "+str(word_no)+" UPDATED IN L1")
            address_list1.deleteNode(address_list1.getnode(address_list1.head,block_no))
            address_list1.push(block_no)

            if block_no in BA2:

                print("ADDRESS FOUND IN CACHE IN L2 ALSO")
                print()
                rint("ADDRESS FOUND AT LINE NO. "+str(block_address2.index(block_no))+" IN L2")
                print()
                print("UPDATING THE GIVEN ADDRESS WITH GIVEN VALUE IN CACHE MEMORY")
                print()
                BA2[block_no][int(block_offset,2)]=[dataa]
                print("WORD NO. "+str(word_no)+" UPDATED IN L2")
                address_list2.deleteNode(address_list2.getnode(address_list2.head,block_no))
                address_list2.push(block_no)

        elif block_no in BA2 and block_no not in BA1 and inptype=='w':

            print("ADDRESS FOUND IN L2")
            print()
            print("LOADING DATA FROM L2 INTO L1")
            print()
            print("LOADING DATA FROM LINE NO. "+str(block_address2.index(block_no))+" IN L2")
            print()
            print("REPLACING DATA IN L1 AT LINE NO. ",end="")

            if len(BA1)<CL:
                print(len(BA1))
                block_address1[len(BA1)]=block_no
            else:
                print(block_address1.index(address_list1.lastNode(address_list1.head).blockno))
                block_address1[block_address1.index(address_list1.lastNode(address_list1.head).blockno)]=block_no
            print()
            print("LOADING BLOCK NO. "+str(int(block_no,2))+" IN L1")
            print()

            print("REPLACED ADDRESS IN L1: ",end="")

            if len(BA1)<CL:

```



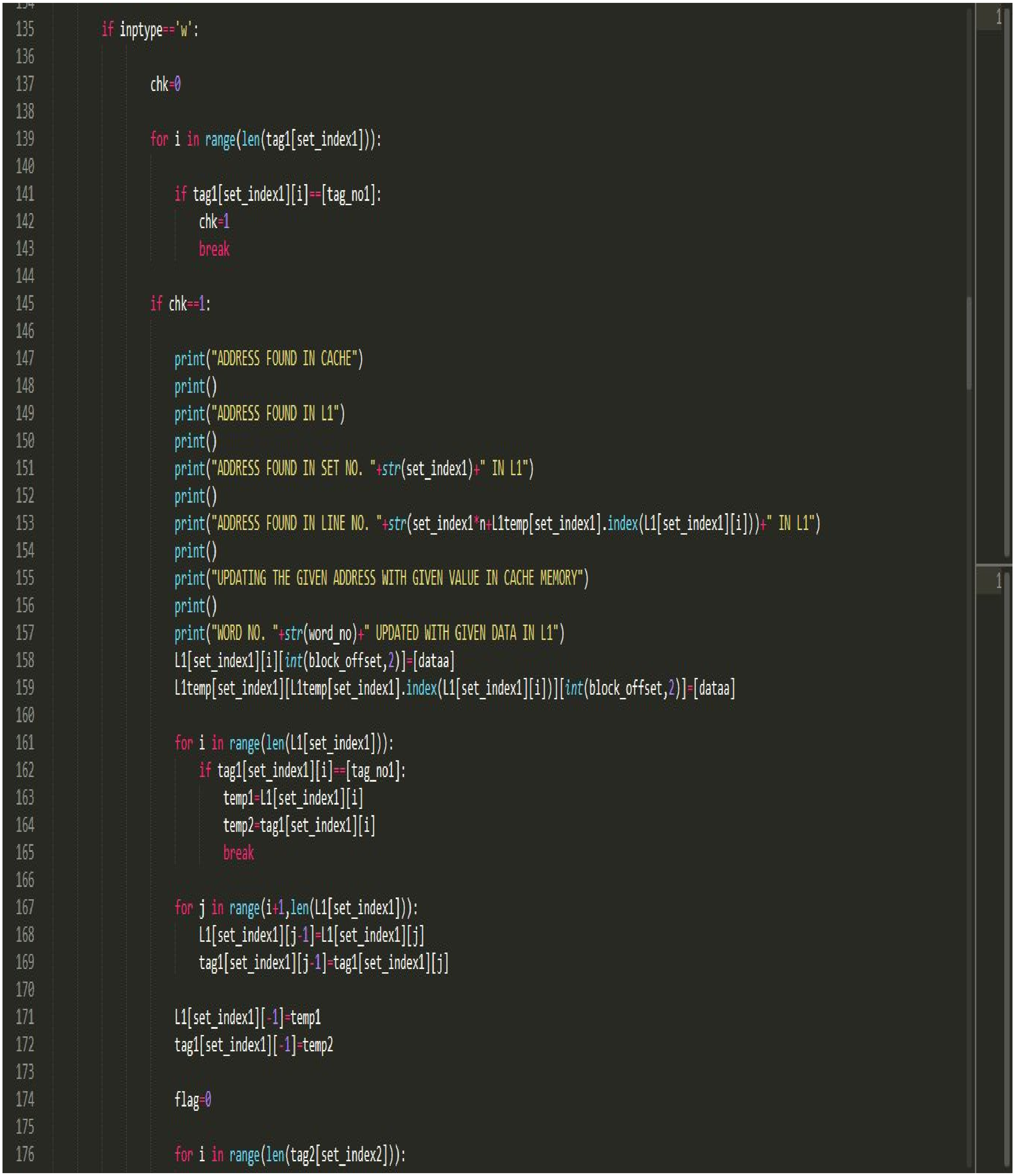
```
                    BA1[block_no]=[["NULL"]]*B
                    address_list1.push(block_no)

                print("REPLACED ADDRESS IN L2: ",end="")

                if len(BA2)<2*CL:

                    BA2[block_no]=[["NULL"]]*B
                    print("Line was empty, no block found!")
                    address_list2.push(block_no)

                else:

                    print("BLOCK NO.: "+address_list2.lastNode(address_list2.head).blockno)
                    del BA2[address_list2.lastNode(address_list2.head).blockno]
                    address_list2.deleteNode(address_list2.lastNode(address_list2.head))
                    BA2[block_no]=[["NULL"]]*B
                    address_list2.push(block_no)

                print()
                print("DATA LOADED IN L1 AND L2")
                print()
                print("UPDATING THE GIVEN ADDRESS WITH GIVEN VALUE IN CACHE MEMORY")
                print()
                print("WORD NO. "+str(word_no)+" UPDATED WITH GIVEN VALUE IN L1 AND L2")
                BA1[block_no][int(block_offset,2)]=[dataa]
                BA2[block_no][int(block_offset,2)]=[dataa]

            else:
                print("CACHE MISS!!!")
                print("ADDRESS NOT IN CACHE MEMORY")

    print()
    L1disp()
    print()
    L2disp()

cont='y'
while cont=='y':
    f()
    print()
    cont=input("continue? (y/n) ")
    print()
```



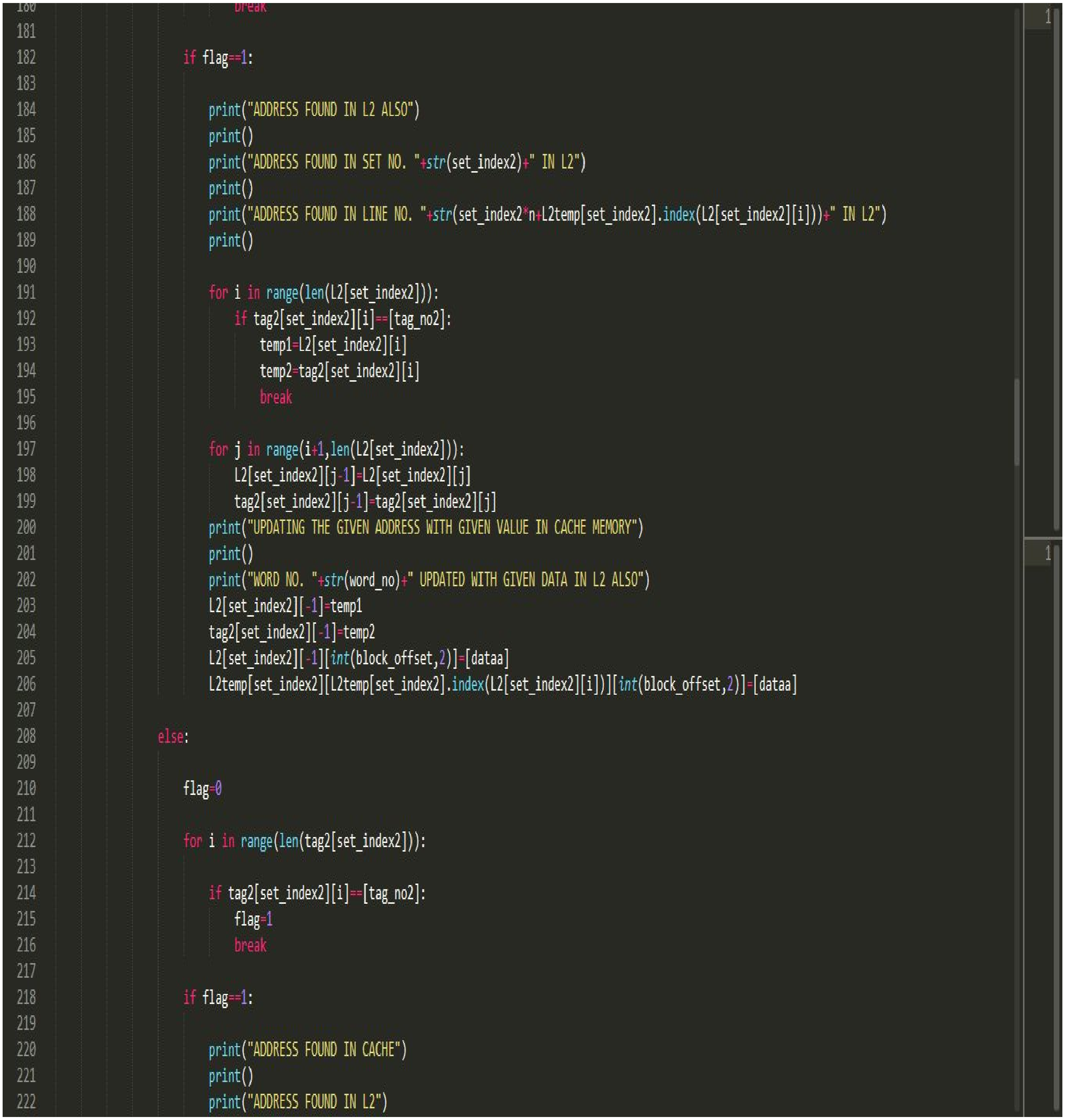
```
print()
print("                         TWO LEVEL CACHE // n WAY SET ASSOCIATIVE MAPPING IMPLEMENTATION                                     ")
print()
print()

N=int(input("Enter main memory size: "))
print()
CL=int(input("Enter number of cache lines: "))
print()
B=int(input("Enter block size: "))
print()
n=int(input("Enter 'n': "))
print()

L1=[]
L1temp=[]
L2=[]
L2temp=[]
tag1=[]
tag1temp=[]
tag2=[]
tag2temp=[]

for i in range((2*CL)//n):
    if i<CL//n:
        L1.append([])
        L1temp.append([])
        tag1.append([])
        tag1temp.append([])
        L2.append([])
        L2temp.append([])
        tag2.append([])
        tag2temp.append([])
    else:
        L2.append([])
        L2temp.append([])
        tag2.append([])
        tag2temp.append([])

for i in range(len(L2)):        # Initialising each line in each set with empty adress in L2
    for j in range(n):
        L2[i].append([["NULL"]]*B)
        L2temp[i].append([["NULL"]]*B)
```



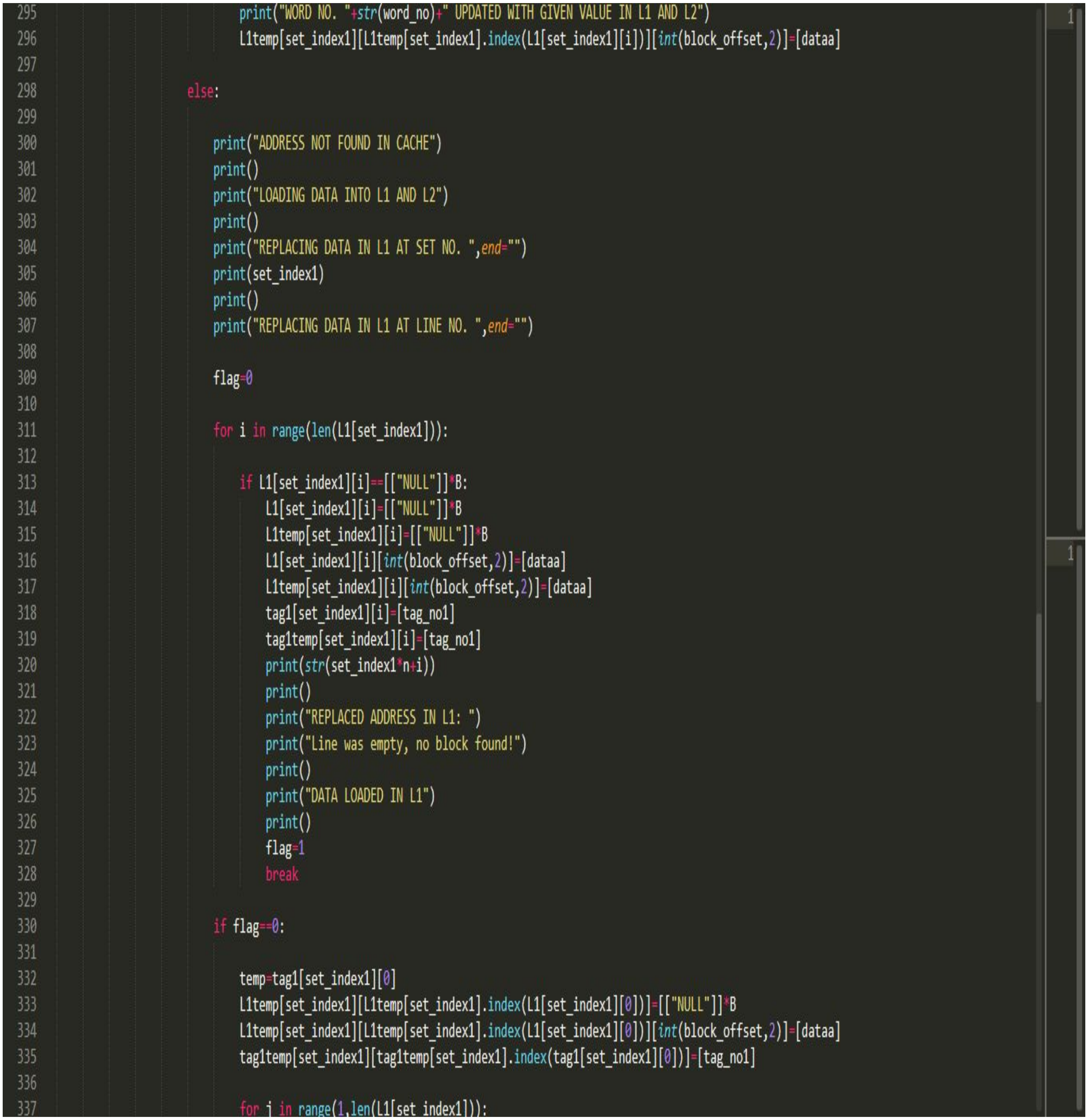
```python

def f():

    inptype=input("write(w)/read(r): ")

    print()

    if inptype=='r':
        add=input("Enter address in binary form: ")

    else:
        add=input("Enter address in binary form: ")
        dataa=input("Enter data: ")

    print()

    if(len(add)!=len(bin(N)[2:])-1):
        print("INVALID ADDRESS!")

    else:

        print("ADDRESS BREAKDOWN: ")
        print()

        word_no=int(add,2)
        print("Word No.: "+str(add)+" ("+str(word_no)+")")

        block_offset=add[len(bin(N)[2:])-len(bin(B)[2:]):]
        print("Block Offset: "+str(block_offset)+" ("+str(int(block_offset,2))+")")

        block_no=add[:len(bin(N)[2:])-len(bin(B)[2:])]
        print("Block no.: "+str(block_no)+" ("+str(int(block_no,2))+")")

        set_no1=block_no[len(block_no)-len(bin(CL//n)[2:])+1:]
        print("Set 1 no.: "+str(set_no1)+" ("+str(int(set_no1,2))+")")

        set_no2=block_no[len(block_no)-len(bin((2*CL)//n)[2:])+1:]
        print("Set 2 no.: "+str(set_no2)+" ("+str(int(set_no2,2))+")")

        tag_no1=block_no[:len(block_no)-len(bin(CL//n)[2:])+1]
        print("Tag1: "+str(tag_no1)+" ("+str(int(tag_no1,2))+")")

        tag_no2=block_no[:len(block_no)-len(bin((2*CL)//n)[2:])+1]
```



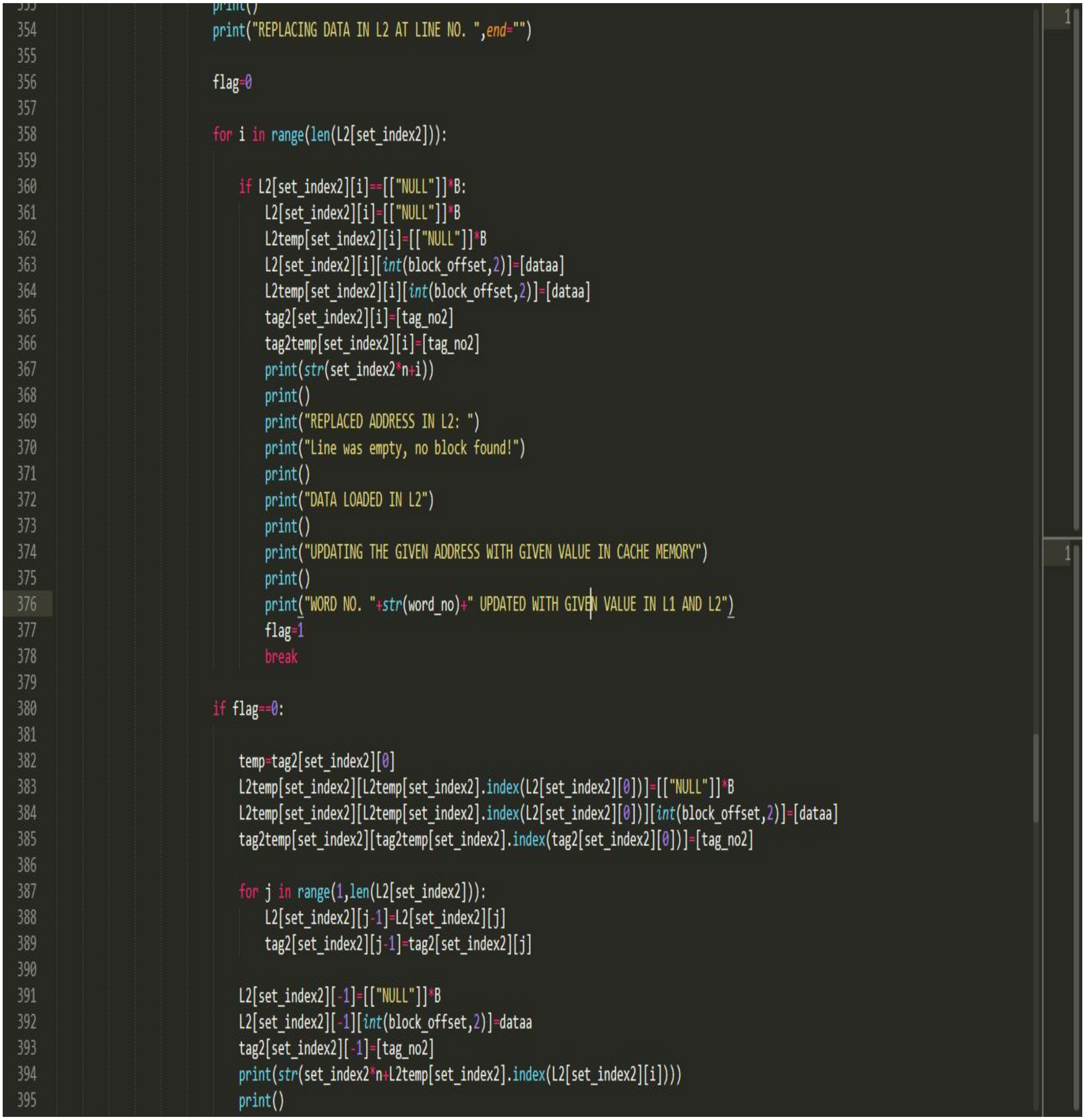
```
        if inptype=='w':

                chk=0

                for i in range(len(tag1[set_index1])):

                    if tag1[set_index1][i]==[tag_no1]:
                        chk=1
                        break

                if chk==1:

                    print("ADDRESS FOUND IN CACHE")
                    print()
                    print("ADDRESS FOUND IN L1")
                    print()
                    print("ADDRESS FOUND IN SET NO. "+str(set_index1)+" IN L1")
                    print()
                    print("ADDRESS FOUND IN LINE NO. "+str(set_index1*n+L1temp[set_index1].index(L1[set_index1][i]))+" IN L1")
                    print()
                    print("UPDATING THE GIVEN ADDRESS WITH GIVEN VALUE IN CACHE MEMORY")
                    print()
                    print("WORD NO. "+str(word_no)+" UPDATED WITH GIVEN DATA IN L1")
                    L1[set_index1][i][int(block_offset,2)]=[dataa]
                    L1temp[set_index1][L1temp[set_index1].index(L1[set_index1][i])][int(block_offset,2)]=[dataa]

                    for i in range(len(L1[set_index1])):
                        if tag1[set_index1][i]==[tag_no1]:
                            temp1=L1[set_index1][i]
                            temp2=tag1[set_index1][i]
                            break

                    for j in range(i+1,len(L1[set_index1])):
                        L1[set_index1][j-1]=L1[set_index1][j]
                        tag1[set_index1][j-1]=tag1[set_index1][j]

                    L1[set_index1][-1]=temp1
                    tag1[set_index1][-1]=temp2

                    flag=0

                    for i in range(len(tag2[set_index2])):
```



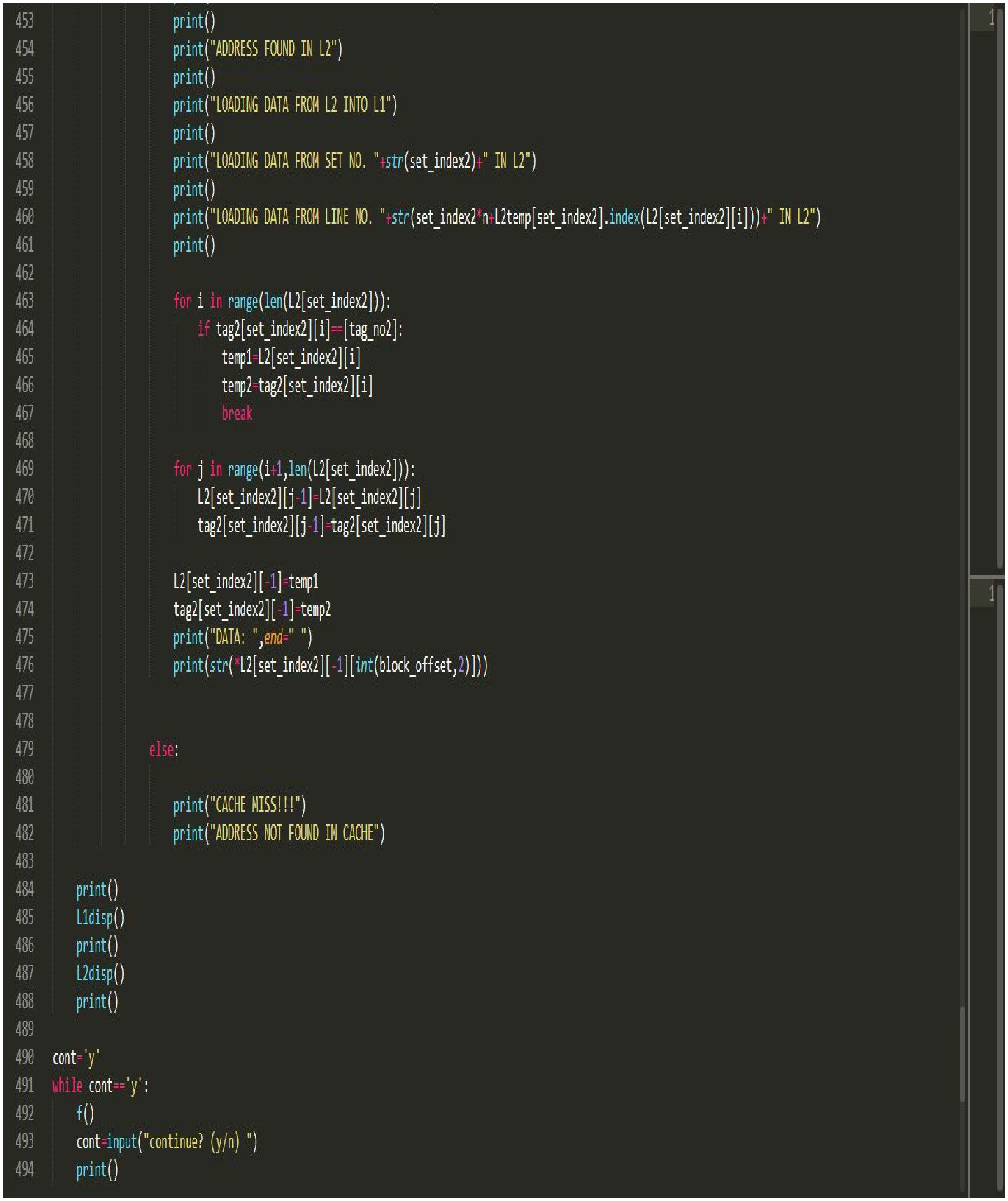
```
                            break

                    if flag==1:

                        print("ADDRESS FOUND IN L2 ALSO")
                        print()
                        print("ADDRESS FOUND IN SET NO. "+str(set_index2)+" IN L2")
                        print()
                        print("ADDRESS FOUND IN LINE NO. "+str(set_index2*n+L2temp[set_index2].index(L2[set_index2][i]))+" IN L2")
                        print()

                        for i in range(len(L2[set_index2])):
                            if tag2[set_index2][i]==[tag_no2]:
                                temp1=L2[set_index2][i]
                                temp2=tag2[set_index2][i]
                                break

                        for j in range(i+1,len(L2[set_index2])):
                            L2[set_index2][j-1]=L2[set_index2][j]
                            tag2[set_index2][j-1]=tag2[set_index2][j]
                        print("UPDATING THE GIVEN ADDRESS WITH GIVEN VALUE IN CACHE MEMORY")
                        print()
                        print("WORD NO. "+str(word_no)+" UPDATED WITH GIVEN DATA IN L2 ALSO")
                        L2[set_index2][-1]=temp1
                        tag2[set_index2][-1]=temp2
                        L2[set_index2][-1][int(block_offset,2)]=[dataa]
                        L2temp[set_index2][L2temp[set_index2].index(L2[set_index2][i])][int(block_offset,2)]=[dataa]

                else:

                    flag=0

                    for i in range(len(tag2[set_index2])):

                        if tag2[set_index2][i]==[tag_no2]:
                            flag=1
                            break

                    if flag==1:

                        print("ADDRESS FOUND IN CACHE")
                        print()
                        print("ADDRESS FOUND IN L2")
```



```
                            print("WORD NO. "+str(word_no)+" UPDATED WITH GIVEN VALUE IN L1 AND L2")
                            L1temp[set_index1][L1temp[set_index1].index(L1[set_index1][i])][int(block_offset,2)]=[dataa]

                    else:

                        print("ADDRESS NOT FOUND IN CACHE")
                        print()
                        print("LOADING DATA INTO L1 AND L2")
                        print()
                        print("REPLACING DATA IN L1 AT SET NO. ",end="")
                        print(set_index1)
                        print()
                        print("REPLACING DATA IN L1 AT LINE NO. ",end="")

                        flag=0

                        for i in range(len(L1[set_index1])):

                            if L1[set_index1][i]==[["NULL"]]*B:
                                L1[set_index1][i]=[["NULL"]]*B
                                L1temp[set_index1][i]=[["NULL"]]*B
                                L1[set_index1][i][int(block_offset,2)]=[dataa]
                                L1temp[set_index1][i][int(block_offset,2)]=[dataa]
                                tag1[set_index1][i]=[tag_no1]
                                tag1temp[set_index1][i]=[tag_no1]
                                print(str(set_index1*n+i))
                                print()
                                print("REPLACED ADDRESS IN L1: ")
                                print("Line was empty, no block found!")
                                print()
                                print("DATA LOADED IN L1")
                                print()
                                flag=1
                                break

                        if flag==0:

                            temp=tag1[set_index1][0]
                            L1temp[set_index1][L1temp[set_index1].index(L1[set_index1][0])]=[["NULL"]]*B
                            L1temp[set_index1][L1temp[set_index1].index(L1[set_index1][0])][int(block_offset,2)]=[dataa]
                            tag1temp[set_index1][tag1temp[set_index1].index(tag1[set_index1][0])]=[tag_no1]

                            for j in range(1,len(L1[set_index1])):
```



```
                        print()
                        print("REPLACING DATA IN L2 AT LINE NO. ",end="")

                        flag=0

                        for i in range(len(L2[set_index2])):

                            if L2[set_index2][i]==[["NULL"]]*B:
                                L2[set_index2][i]=[["NULL"]]*B
                                L2temp[set_index2][i]=[["NULL"]]*B
                                L2[set_index2][i][int(block_offset,2)]=[dataa]
                                L2temp[set_index2][i][int(block_offset,2)]=[dataa]
                                tag2[set_index2][i]=[tag_no2]
                                tag2temp[set_index2][i]=[tag_no2]
                                print(str(set_index2*n+i))
                                print()
                                print("REPLACED ADDRESS IN L2: ")
                                print("Line was empty, no block found!")
                                print()
                                print("DATA LOADED IN L2")
                                print()
                                print("UPDATING THE GIVEN ADDRESS WITH GIVEN VALUE IN CACHE MEMORY")
                                print()
                                print("WORD NO. "+str(word_no)+" UPDATED WITH GIVEN VALUE IN L1 AND L2")
                                flag=1
                                break

                        if flag==0:

                            temp=tag2[set_index2][0]
                            L2temp[set_index2][L2temp[set_index2].index(L2[set_index2][0])]=[["NULL"]]*B
                            L2temp[set_index2][L2temp[set_index2].index(L2[set_index2][0])][int(block_offset,2)]=[dataa]
                            tag2temp[set_index2][tag2temp[set_index2].index(tag2[set_index2][0])]=[tag_no2]

                            for j in range(1,len(L2[set_index2])):
                                L2[set_index2][j-1]=L2[set_index2][j]
                                tag2[set_index2][j-1]=tag2[set_index2][j]

                            L2[set_index2][-1]=[["NULL"]]*B
                            L2[set_index2][-1][int(block_offset,2)]=dataa
                            tag2[set_index2][-1]=[tag_no2]
                            print(str(set_index2*n+L2temp[set_index2].index(L2[set_index2][i])))
                            print()
```



```
                    print()
                    print("ADDRESS FOUND IN L2")
                    print()
                    print("LOADING DATA FROM L2 INTO L1")
                    print()
                    print("LOADING DATA FROM SET NO. "+str(set_index2)+" IN L2")
                    print()
                    print("LOADING DATA FROM LINE NO. "+str(set_index2*n+L2temp[set_index2].index(L2[set_index2][i]))+" IN L2")
                    print()

                    for i in range(len(L2[set_index2])):
                        if tag2[set_index2][i]==[tag_no2]:
                            temp1=L2[set_index2][i]
                            temp2=tag2[set_index2][i]
                            break

                    for j in range(i+1,len(L2[set_index2])):
                        L2[set_index2][j-1]=L2[set_index2][j]
                        tag2[set_index2][j-1]=tag2[set_index2][j]

                    L2[set_index2][-1]=temp1
                    tag2[set_index2][-1]=temp2
                    print("DATA: ",end=" ")
                    print(str(*L2[set_index2][-1][int(block_offset,2)]))


                else:

                    print("CACHE MISS!!!")
                    print("ADDRESS NOT FOUND IN CACHE")

    print()
    L1disp()
    print()
    L2disp()
    print()

cont='y'
while cont=='y':
    f()
    cont=input("continue? (y/n) ")
    print()
```