Documentation

Final Project - Cache memory mapping techniques

Course : CSE112 Computer Organization

Project members: Vaibhav Gupta :- 2019341

Assumptions:

- Even though not asked by the project guidelines, I have maintained a Main memory.
- Each block contains a certain number of words. These words are not further divided into smaller memory bytes and bits.
- Word number represents the data placed at that memory address.
- When Cache is created, it already contains blocks according to the mapping selected. User will not load data into the Cache.
- When the user searches for an address, they will have to enter the integer value of the word to be searched in the Cache.

Cache memory:

It is a special, very high-speed memory. It is used to speed up and synchronize with a high-speed CPU. Cache memory is costlier than Main/Disk memory but cheaper than CPU registers. Cache memory is an extremely fast memory that acts as a buffer between CPU and the RAM. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

It is used to reduce the average time to access data from the Main Memory. The Cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations.

Direct mapping:

This is the simplest technique of all the three. In this technique, each block of Main memory can only be stored in one Cache line. If a line already holds a memory block when an uncontained block is requested, that block is trashed and the requested one is loaded. This type of mapping is very rigid and does not provide any flexibility although it is the cheapest application among the three.

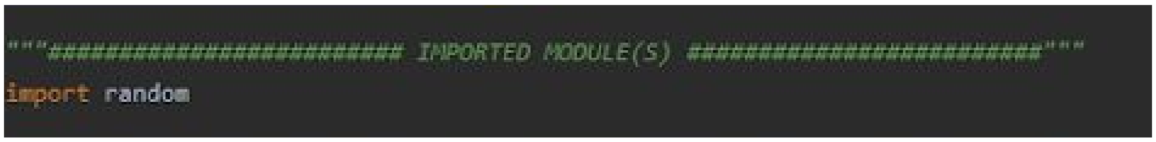
Fully Associative mapping:

In this mapping technique, the blocks are not assigned any specific Cache line. A Main memory block can be copied to any Cache line. This mapping gives us a lot of flexibility but it is very expensive as at any point in time the whole Cache might need to be searched as the memory block could be anywhere in the Cache lines. Parallel searching is done by the CAM word which is a very expensive piece of hardware.

N-way Set associative mapping:

This type of mapping gives us more flexibility than direct mapping but is not as flexible as Fully associative mapping. Instead of allocating a single Cache line for a Main memory block to be copied to, this mapping groups a few Cache lines together thus forming a Set. So, now the Main memory block can be copied to one of these blocks as compared to only one. But the same memory block cannot be copied to any other Cache line outside this set.

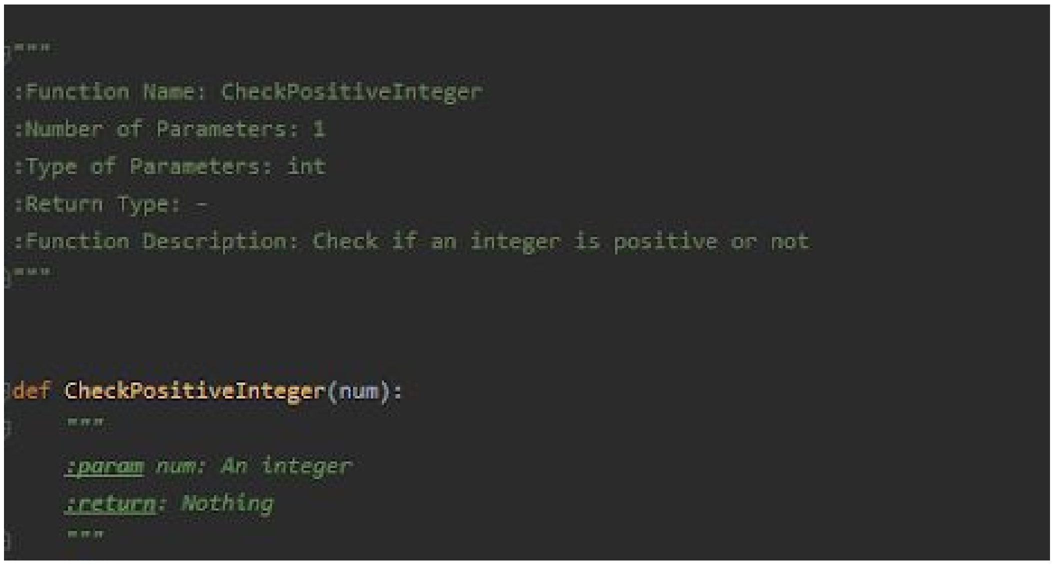
Imported module:



```
"""########################## IMPORTED MODULE(S) ##########################"""
import random
```

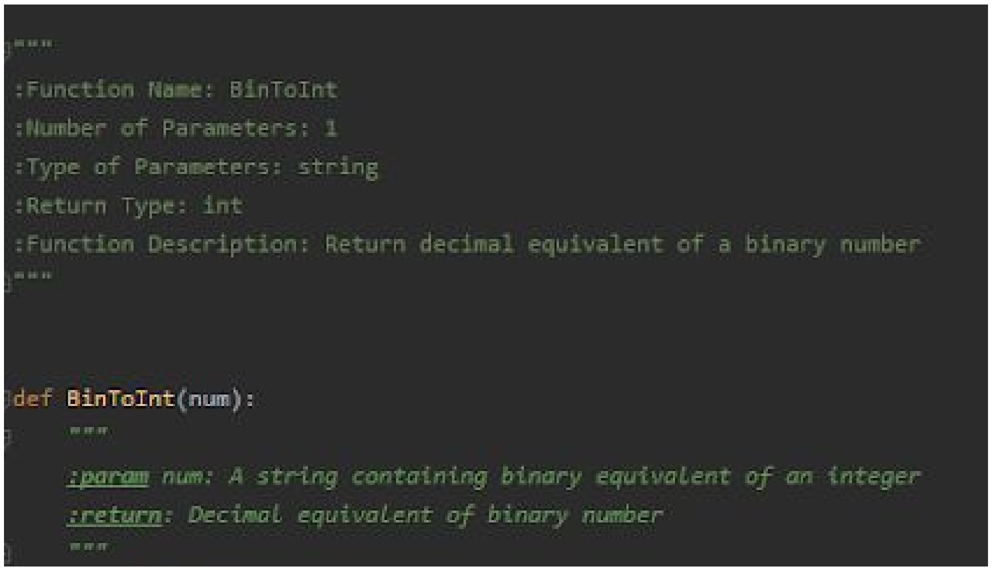
To generate a random number or sets of random numbers in certain places

Methods & Working:



```
"""
:Function Name: CheckPositiveInteger
:Number of Parameters: 1
:Type of Parameters: int
:Return Type: -
:Function Description: Check if an integer is positive or not
"""


def CheckPositiveInteger(num):
    """
    :param num: An integer
    :return: Nothing
    """
```

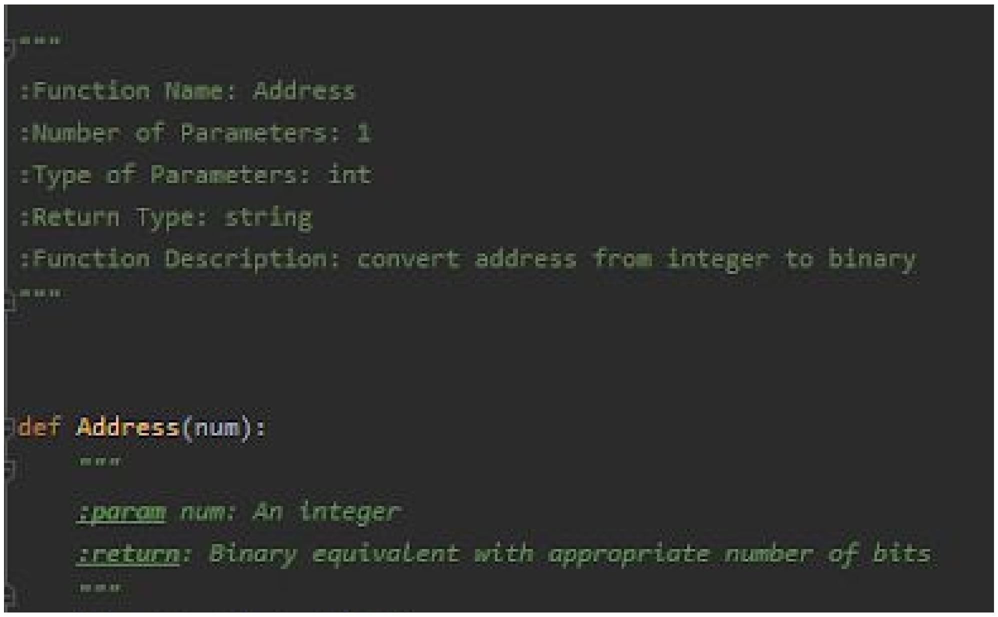


```
"""
:Function Name: BinToInt
:Number of Parameters: 1
:Type of Parameters: string
:Return Type: int
:Function Description: Return decimal equivalent of a binary number
"""


def BinToInt(num):
    """
    :param num: A string containing binary equivalent of an integer
    :return: Decimal equivalent of binary number
    """
```

Eg:

```
>>    print(BinToInt(“1111”))
>>    15
```



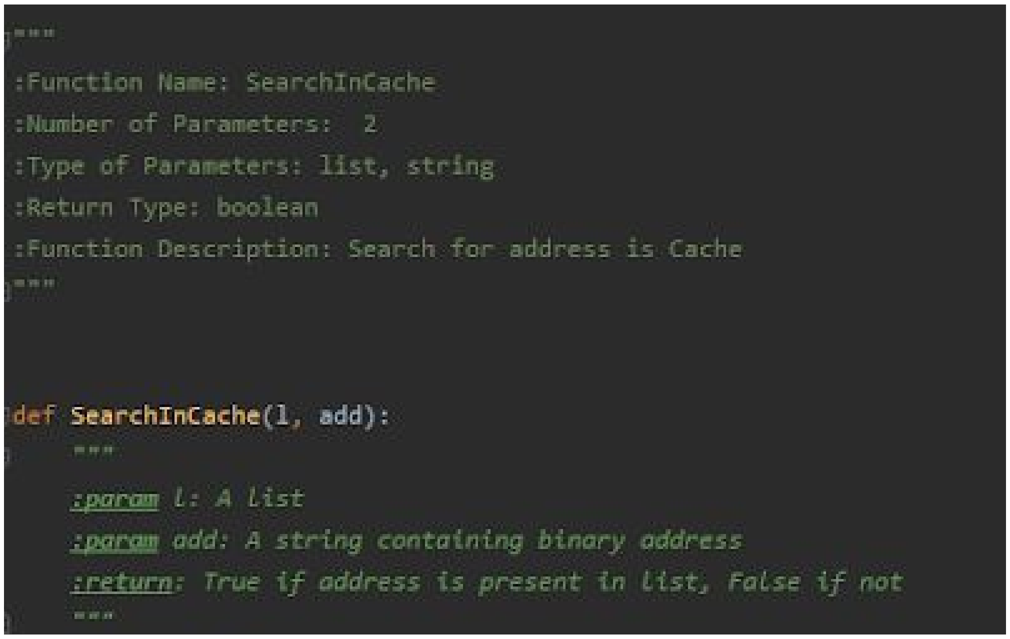
```
"""
:Function Name: Address
:Number of Parameters: 1
:Type of Parameters: int
:Return Type: string
:Function Description: convert address from integer to binary
"""


def Address(num):
    """
    :param num: An integer
    :return: Binary equivalent with appropriate number of bits
    """
```

Eg:

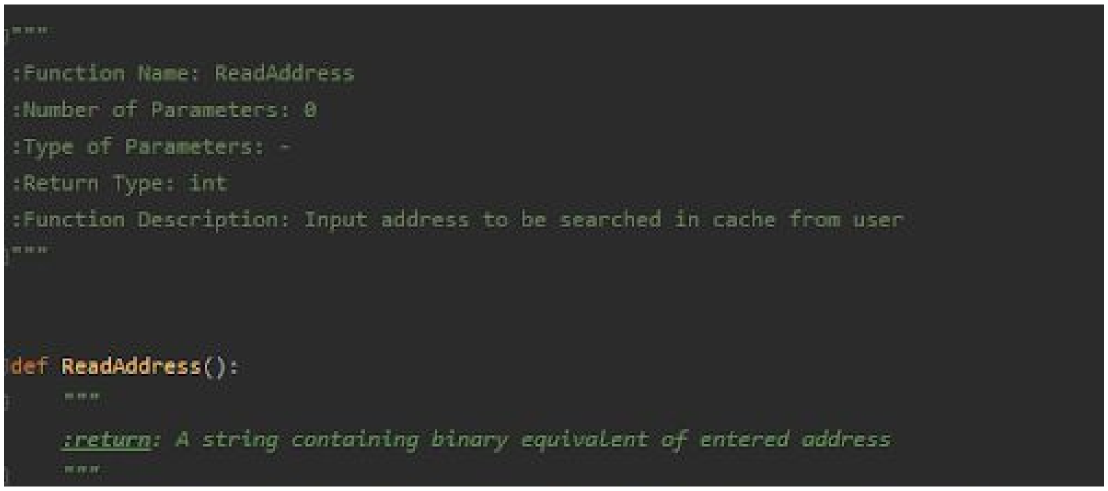
(for a 6 bit address)

```
>>   print(Address(15))
>>   '001111'
```



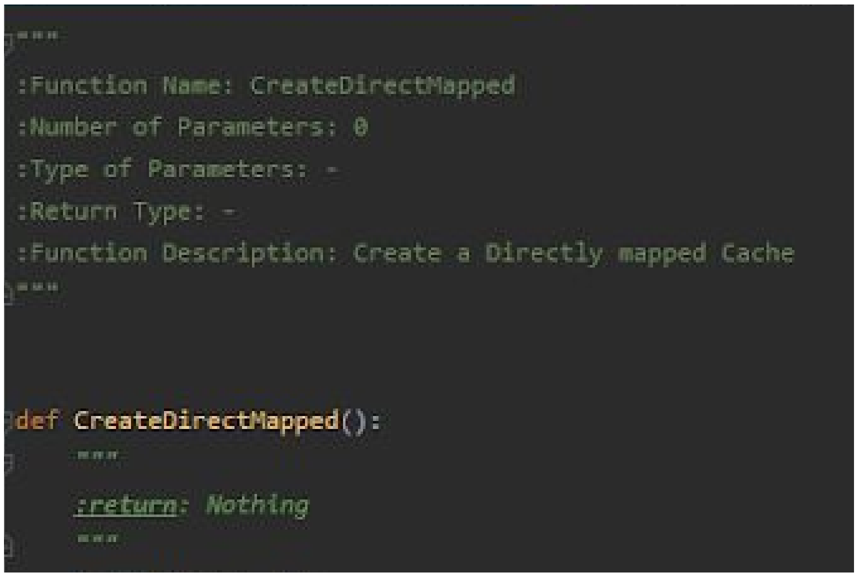
```
"""
:Function Name: SearchInCache
:Number of Parameters:  2
:Type of Parameters: list, string
:Return Type: boolean
:Function Description: Search for address is Cache
"""


def SearchInCache(l, add):
    """
    :param l: A list
    :param add: A string containing binary address
    :return: True if address is present in list, False if not
    """
```



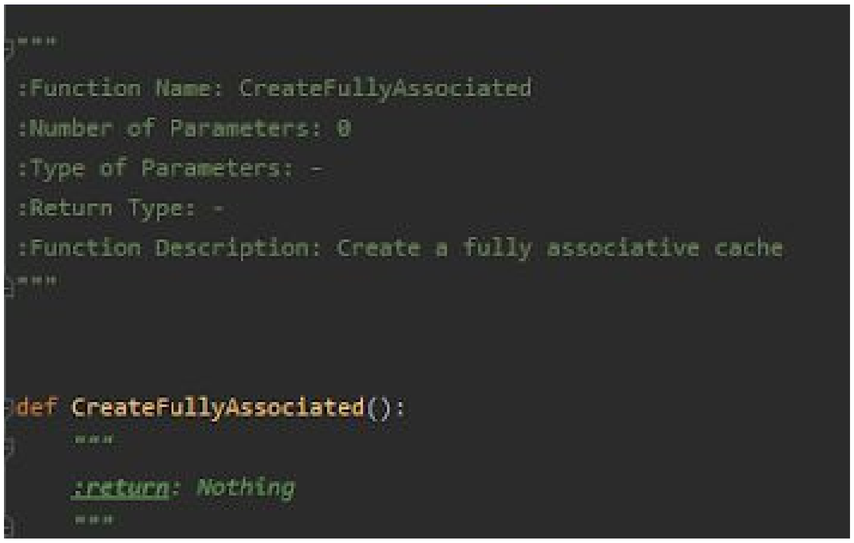
```
"""
:Function Name: ReadAddress
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: int
:Function Description: Input address to be searched in cache from user
"""


def ReadAddress():
    """
    :return: A string containing binary equivalent of entered address
    """
```



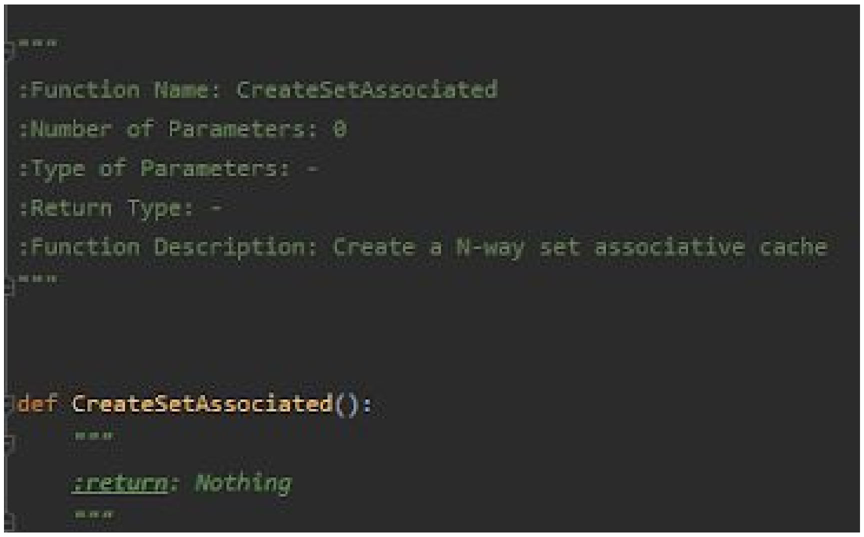
```
"""
:Function Name: CreateDirectMapped
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: Create a Directly mapped Cache
"""


def CreateDirectMapped():
    """
    :return: Nothing
    """
```



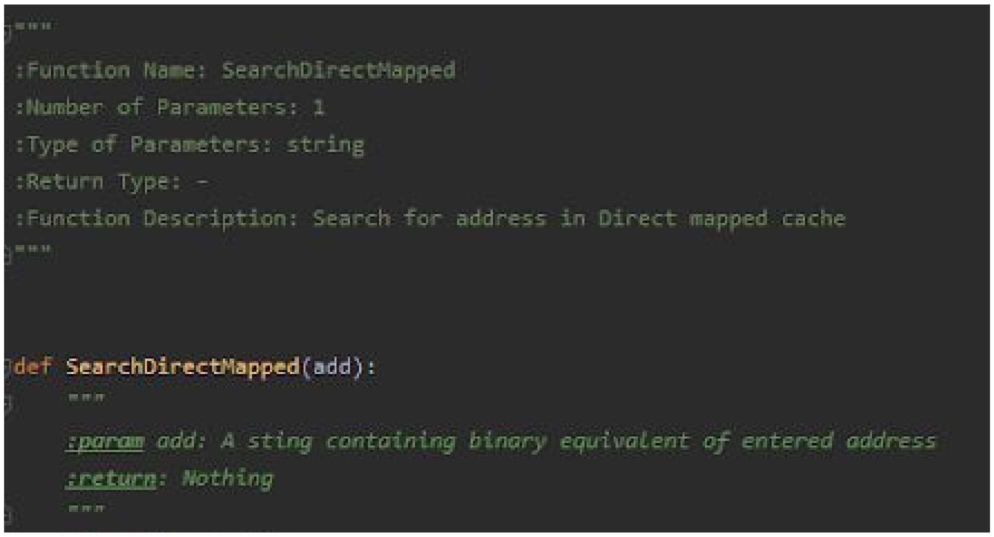
```
"""
:Function Name: CreateFullyAssociated
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: Create a fully associative cache
"""


def CreateFullyAssociated():
    """
    :return: Nothing
    """
```



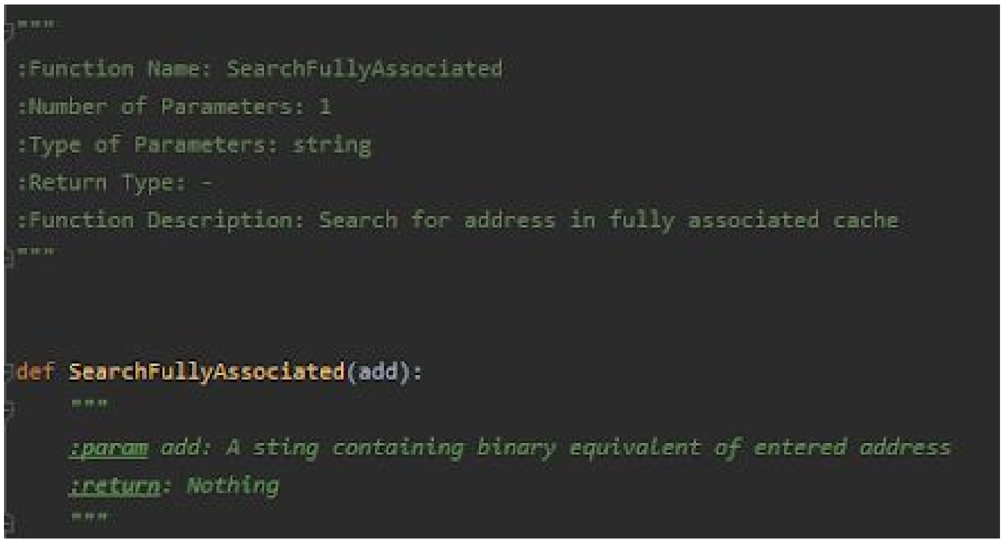
```python
"""
:Function Name: CreateSetAssociated
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: Create a N-way set associative cache
"""


def CreateSetAssociated():
    """
    :return: Nothing
    """
```



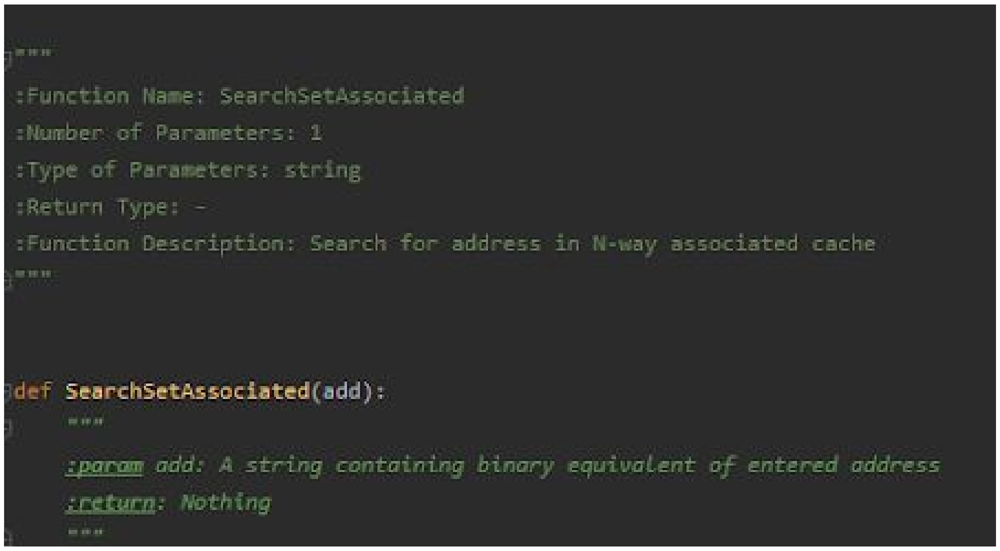
```python
"""
:Function Name: SearchDirectMapped
:Number of Parameters: 1
:Type of Parameters: string
:Return Type: -
:Function Description: Search for address in Direct mapped cache
"""


def SearchDirectMapped(add):
    """
    :param add: A sting containing binary equivalent of entered address
    :return: Nothing
    """
```



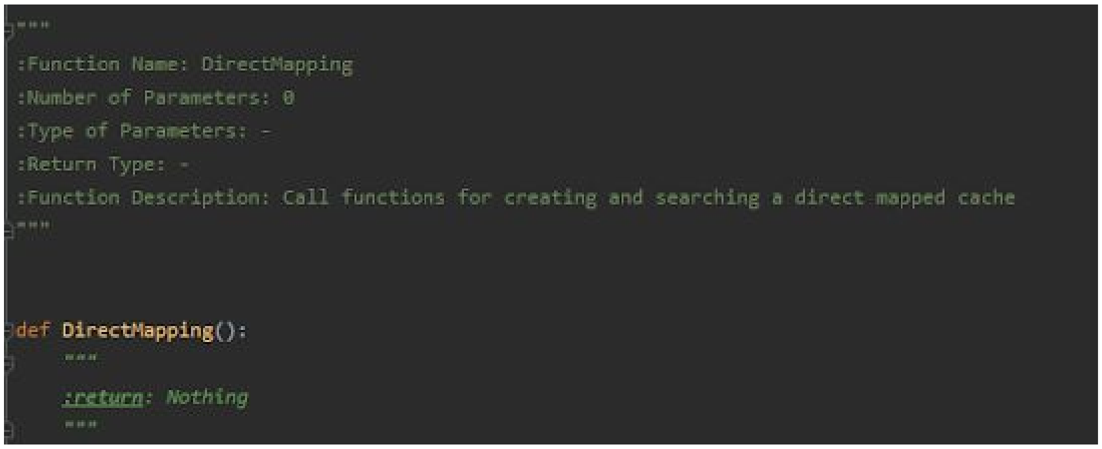
```python
"""
:Function Name: SearchFullyAssociated
:Number of Parameters: 1
:Type of Parameters: string
:Return Type: -
:Function Description: Search for address in fully associated cache
"""


def SearchFullyAssociated(add):
    """
    :param add: A sting containing binary equivalent of entered address
    :return: Nothing
    """
```



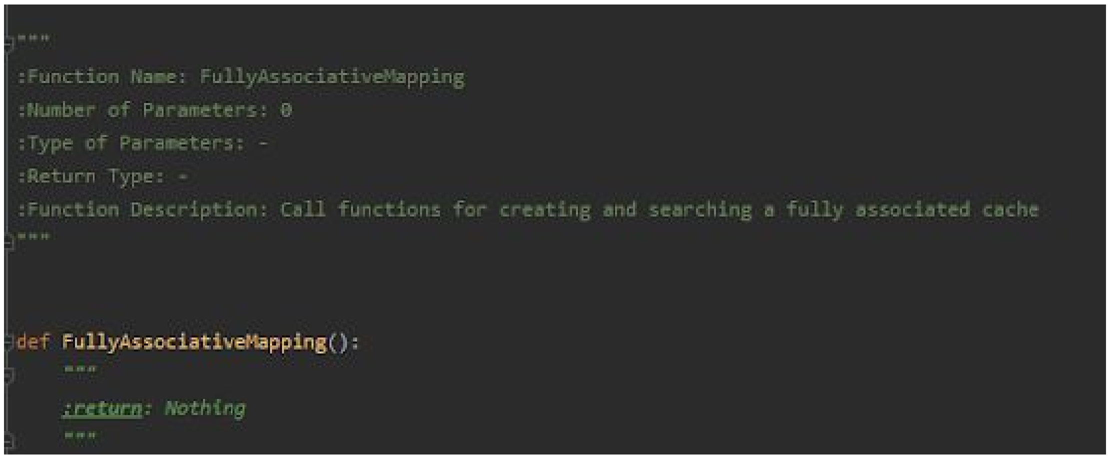
```
"""
:Function Name: SearchSetAssociated
:Number of Parameters: 1
:Type of Parameters: string
:Return Type: -
:Function Description: Search for address in N-way associated cache
"""


def SearchSetAssociated(add):
    """
    :param add: A string containing binary equivalent of entered address
    :return: Nothing
    """
```



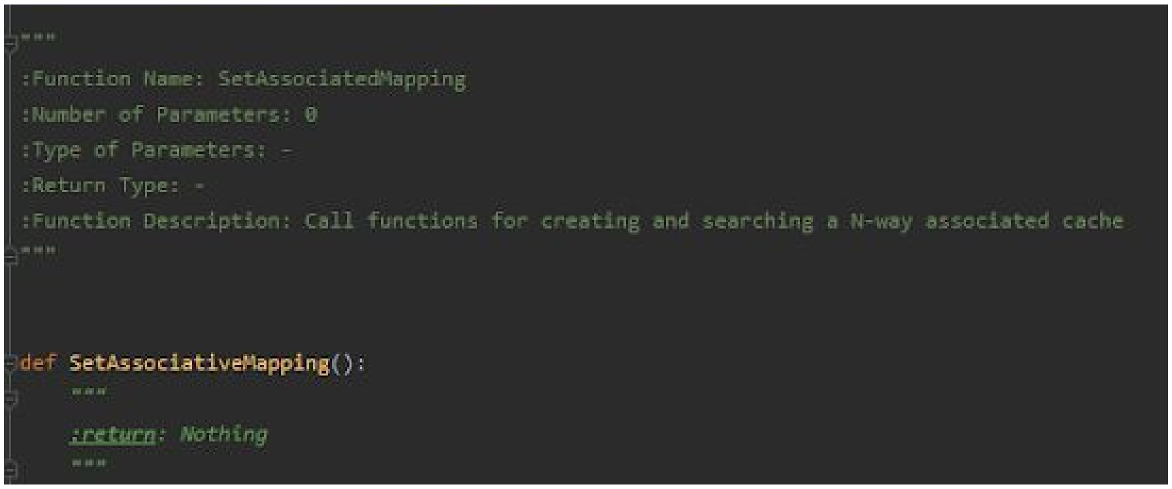
```
"""
:Function Name: DirectMapping
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: Call functions for creating and searching a direct mapped cache
"""


def DirectMapping():
    """
    :return: Nothing
    """
```



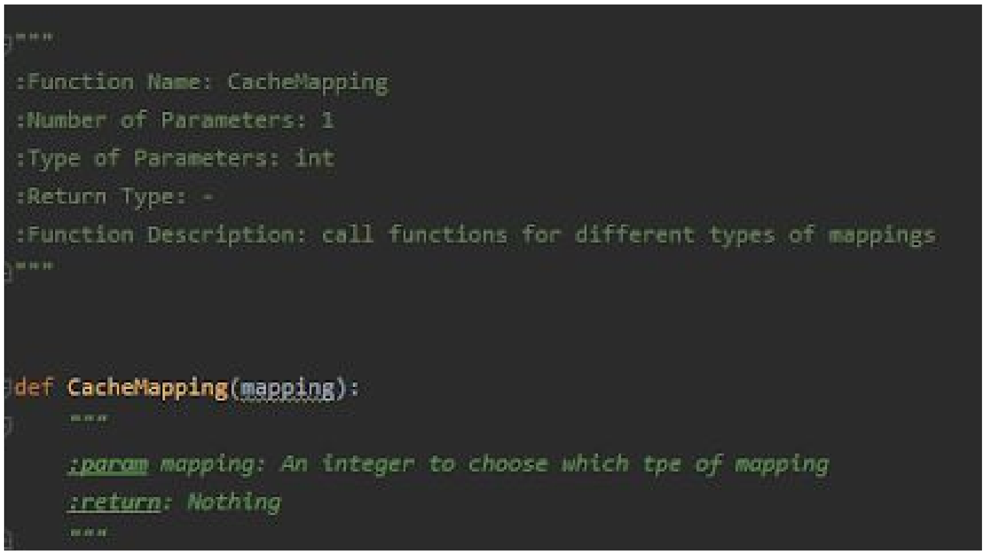
```
"""
:Function Name: FullyAssociativeMapping
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: Call functions for creating and searching a fully associated cache
"""


def FullyAssociativeMapping():
    """
    :return: Nothing
    """
```



```
"""
:Function Name: SetAssociatedMapping
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: Call functions for creating and searching a N-way associated cache
"""


def SetAssociativeMapping():
    """
    :return: Nothing
    """
```



```
"""
:Function Name: CacheMapping
:Number of Parameters: 1
:Type of Parameters: int
:Return Type: -
:Function Description: call functions for different types of mappings
"""


def CacheMapping(mapping):
    """
    :param mapping: An integer to choose which tpe of mapping
    :return: Nothing
    """
```

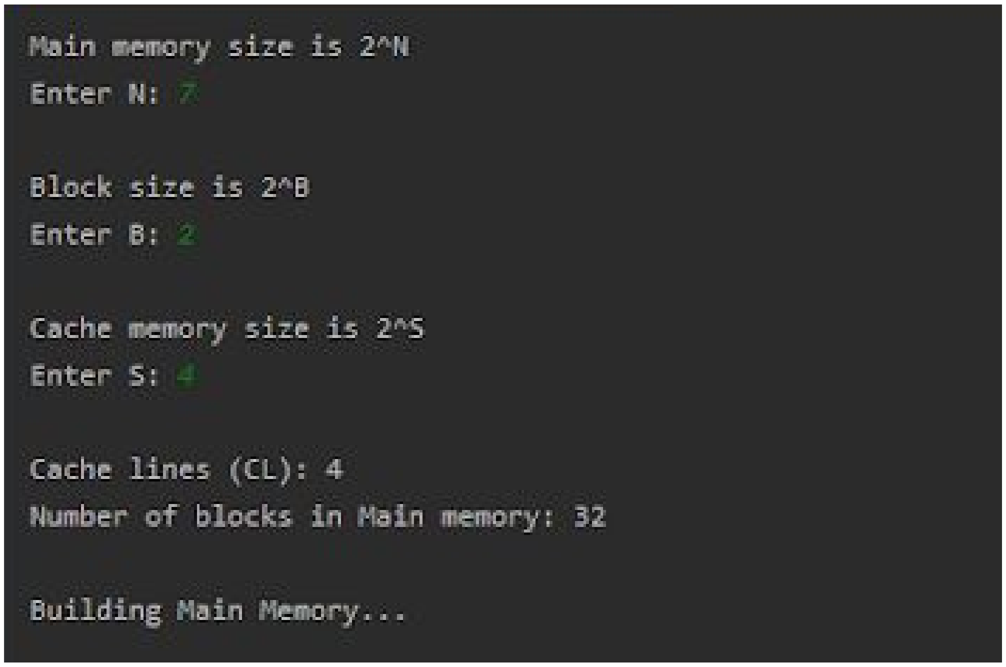
Sample Inputs & Outputs:

Common example for all three types of mapping:

Size of memory : 2^N
-> N = 7

Block size : 2^B
-> B = 2

Size of cache : 2^S
-> S = 4



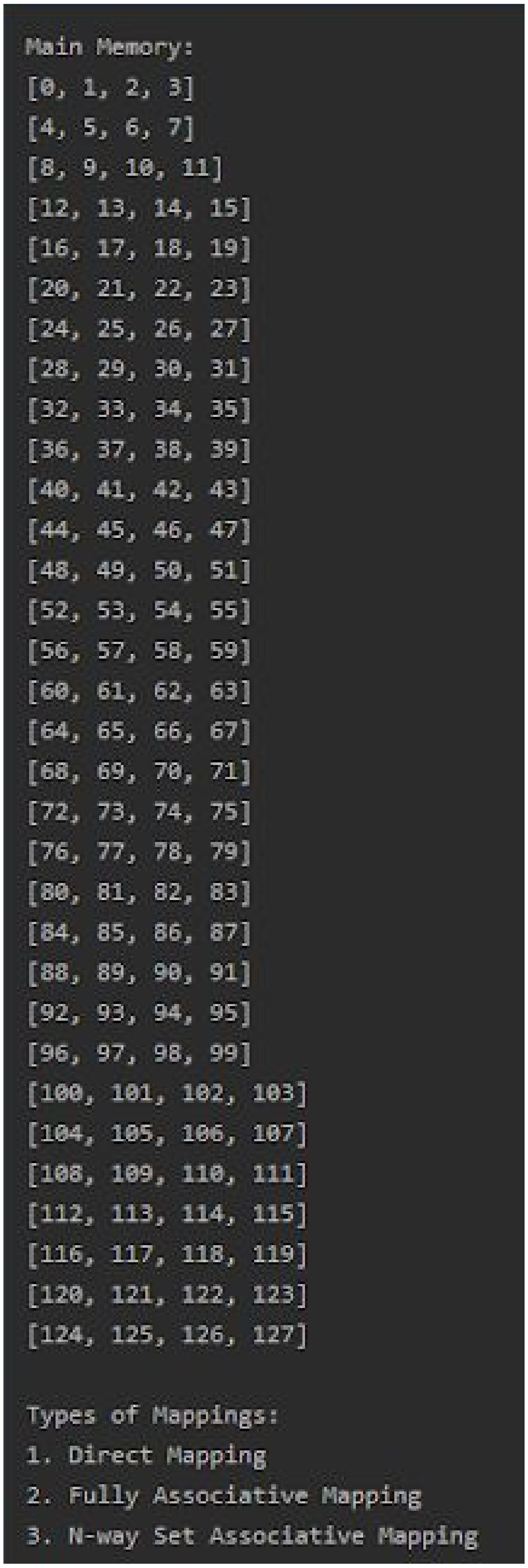
```
Main memory size is 2^N
Enter N: 7

Block size is 2^B
Enter B: 2

Cache memory size is 2^S
Enter S: 4

Cache lines (CL): 4
Number of blocks in Main memory: 32

Building Main Memory...
```



```
Main Memory:
[0, 1, 2, 3]
[4, 5, 6, 7]
[8, 9, 10, 11]
[12, 13, 14, 15]
[16, 17, 18, 19]
[20, 21, 22, 23]
[24, 25, 26, 27]
[28, 29, 30, 31]
[32, 33, 34, 35]
[36, 37, 38, 39]
[40, 41, 42, 43]
[44, 45, 46, 47]
[48, 49, 50, 51]
[52, 53, 54, 55]
[56, 57, 58, 59]
[60, 61, 62, 63]
[64, 65, 66, 67]
[68, 69, 70, 71]
[72, 73, 74, 75]
[76, 77, 78, 79]
[80, 81, 82, 83]
[84, 85, 86, 87]
[88, 89, 90, 91]
[92, 93, 94, 95]
[96, 97, 98, 99]
[100, 101, 102, 103]
[104, 105, 106, 107]
[108, 109, 110, 111]
[112, 113, 114, 115]
[116, 117, 118, 119]
[120, 121, 122, 123]
[124, 125, 126, 127]

Types of Mappings:
1. Direct Mapping
2. Fully Associative Mapping
3. N-way Set Associative Mapping
```

Direct Mapping:

Cache Lines	Main memory blocks
0	0, 4, 8, 12, 16, 20, 24, 28
1	1, 5, 9, 13, 17, 21, 25, 29
2	2, 6, 10, 14, 18, 22, 26, 30
3	3, 7, 11, 15, 19, 23, 27, 31

In the following example:

Address requested = 66 => 1000010

Since, B = 2 therefore 2 least significant bits represent the block offset i.e. 10

The remaining portion is divided into 2 parts, the tag and line.

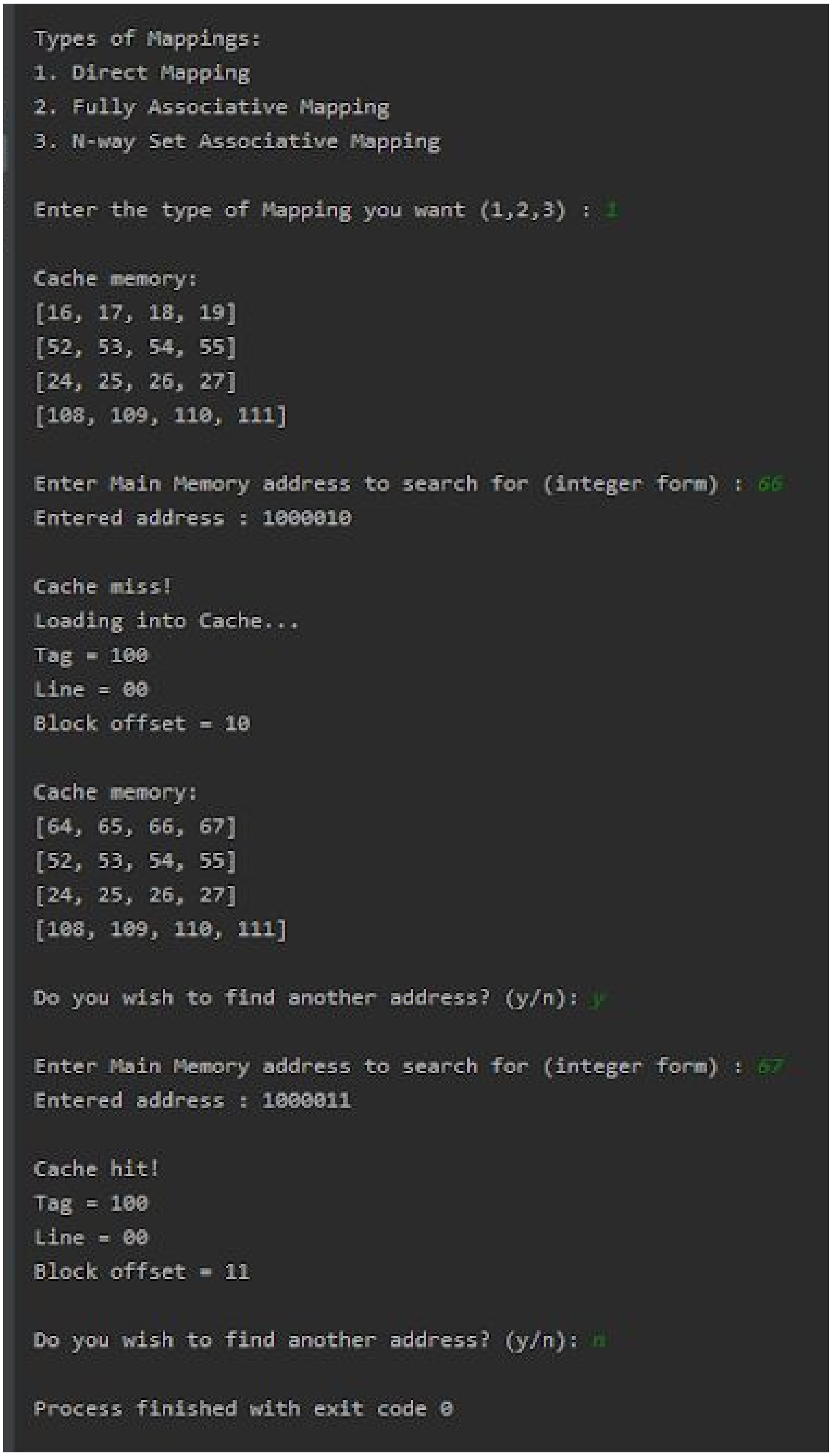
Since, no. of cache lines = 2^(S-B) = 2^CL = 4. CL = 2

Hence the next 2 least significant bits represent the line number in which the block which holds 66 can be loaded.

The remaining bits represent the tag. In this case 100. I.e. the fourth block allotted to the 0th line. I.e. the 16th Main memory block.

And referring to the above image, the 2nd (10) offset of that block is 66.

Next 67 has been requested to show locality of reference.



```
Types of Mappings:
1. Direct Mapping
2. Fully Associative Mapping
3. N-way Set Associative Mapping

Enter the type of Mapping you want (1,2,3) : 1

Cache memory:
[16, 17, 18, 19]
[52, 53, 54, 55]
[24, 25, 26, 27]
[108, 109, 110, 111]

Enter Main Memory address to search for (integer form) : 66
Entered address : 1000010

Cache miss!
Loading into Cache...
Tag = 100
Line = 00
Block offset = 10

Cache memory:
[64, 65, 66, 67]
[52, 53, 54, 55]
[24, 25, 26, 27]
[108, 109, 110, 111]

Do you wish to find another address? (y/n): y

Enter Main Memory address to search for (integer form) : 67
Entered address : 1000011

Cache hit!
Tag = 100
Line = 00
Block offset = 11

Do you wish to find another address? (y/n): n

Process finished with exit code 0
```

Fully Associative Mapping:

Any block can be loaded into any Cache line.

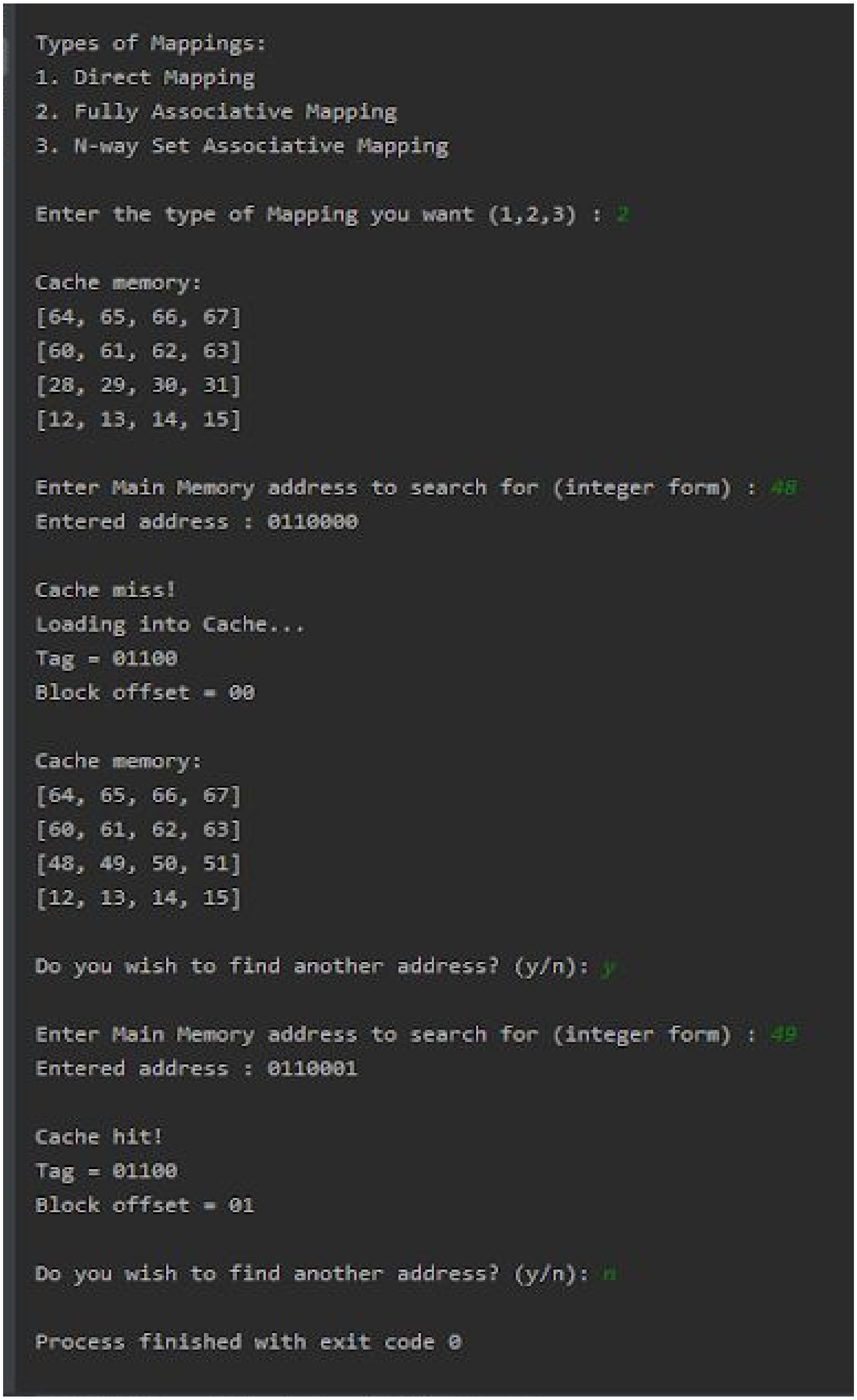
In the following example:

Address requested = 48
=> 0110000

Since, B = 2 therefore 2 least significant bits represent the block offset i.e. 00

The remaining portion is the tag i.e. 01100

Next 49 has been requested to show locality of reference



```
Types of Mappings:
1. Direct Mapping
2. Fully Associative Mapping
3. N-way Set Associative Mapping

Enter the type of Mapping you want (1,2,3) : 2

Cache memory:
[64, 65, 66, 67]
[60, 61, 62, 63]
[28, 29, 30, 31]
[12, 13, 14, 15]

Enter Main Memory address to search for (integer form) : 48
Entered address : 0110000

Cache miss!
Loading into Cache...
Tag = 01100
Block offset = 00

Cache memory:
[64, 65, 66, 67]
[60, 61, 62, 63]
[48, 49, 50, 51]
[12, 13, 14, 15]

Do you wish to find another address? (y/n): y

Enter Main Memory address to search for (integer form) : 49
Entered address : 0110001

Cache hit!
Tag = 01100
Block offset = 01

Do you wish to find another address? (y/n): n

Process finished with exit code 0
```

N-way Set Associative Mapping

Set number	Cache line
0	0, 1
1	2, 3

Set number	Main memory blocks
0	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
1	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31

Each set can hold two blocks from the allotted ones

Requested address = 96 => 1100000

Since, B = 2 therefore 2 least significant bits represent the block offset i.e. 00

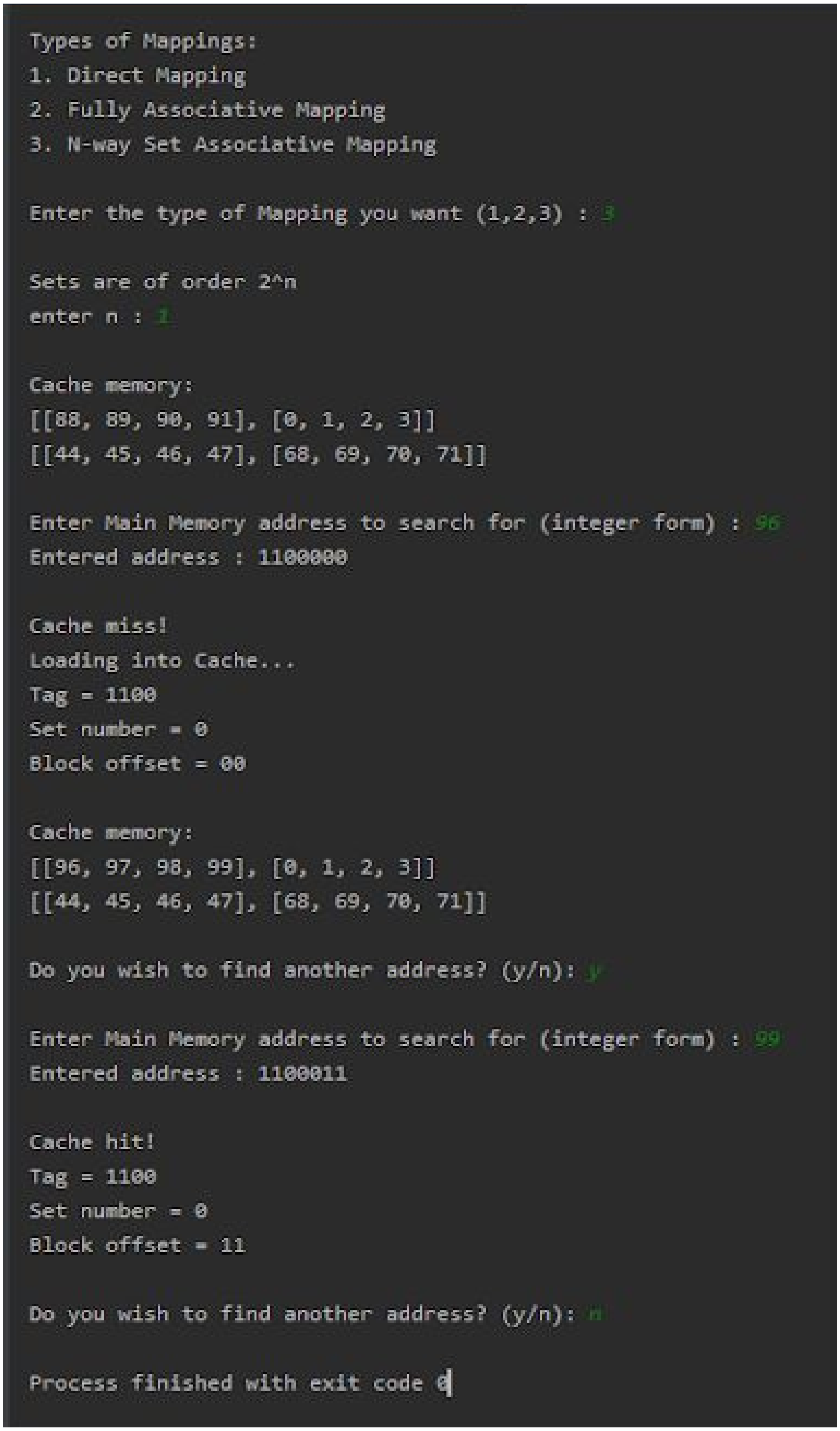
The rest of the address will be divided into two parts i.e. tag and set number

Set size = 2^n = 2. Therefore n = 1

So the next least significant bit is the set number i.e. 0

The rest is the tag i.e. 1100

Next 99 has been requested to show locality of reference



```
Types of Mappings:
1. Direct Mapping
2. Fully Associative Mapping
3. N-way Set Associative Mapping

Enter the type of Mapping you want (1,2,3) : 3

Sets are of order 2^n
enter n : 1

Cache memory:
[[88, 89, 90, 91], [0, 1, 2, 3]]
[[44, 45, 46, 47], [68, 69, 70, 71]]

Enter Main Memory address to search for (integer form) : 96
Entered address : 1100000

Cache miss!
Loading into Cache...
Tag = 1100
Set number = 0
Block offset = 00

Cache memory:
[[96, 97, 98, 99], [0, 1, 2, 3]]
[[44, 45, 46, 47], [68, 69, 70, 71]]

Do you wish to find another address? (y/n): y

Enter Main Memory address to search for (integer form) : 99
Entered address : 1100011

Cache hit!
Tag = 1100
Set number = 0
Block offset = 11

Do you wish to find another address? (y/n): n

Process finished with exit code 0
```