German University in Cairo
Faculty of Media Engineering and Technology
Dr. Nada Sharaf

Concepts of Programming Languages, Spring 2021
CPU Cache: Prolog Predicates and Examples
Deadline: 10 June 2021

Representations

Cache

The cache is represented by list of items. The item is a structure that is defined as follows:
`item(Tag,Data,ValidBit,Order)`

- `Tag` is a structure `tag(StringTag)` where `StringTag` is a string of the binary number which represents the tag discussed previously.
- `Data` is represented by the structure `data(MemData)`
- `ValidBit` is the validity bit where zero means the data is not valid (trash data) and one meaning it is valid and we can retrieve it.
- `Order` is the decimal number representing the order of the placement of the item in cache. The lower the number, the newer it is replaced with zero being the least number.

An example of a cache is shown below:

```
[item(tag("00000"),data(1),0,0), item(tag("00000"),data(b),1,0),
item(tag("00010"),data(0),0,3), item(tag("00000"),data(c),0,2)]
```

This example represents the cache:

Block Index	Tag	Valid	Data
000	00000	0	1
001	00000	1	b
010	00010	0	0
011	00000	0	c

Memory

The memory is represented as a list and the order inside it represent the address meaning the first element in the list is of address zero.
e.g. `[0,a,b,1,1,c,d,0,1,1,0,0,1,1,0,f]`

German University in Cairo
Faculty of Media Engineering and Technology
Dr. Nada Sharaf

Helpful predefined predicates

Hints:

`atom_number(Atom, Number)` Bi-directional conversion between atom and number. At least one of the two arguments must be instantiated.

To concatenate strings `string_concat(String1, String2, String3)`
As append does with lists, string_concate concatenate the end of String1 with String2 in String3.

Predicates to be added

You are going to implement this system purely through Prolog, you can add as many predicates as you need to make sure the following predicates work correctly. Your solution must utilise both techniques, unification and generate-and-test. You **have to implement ALL of the following predicates. Your implementation should be GENERIC meaning it accepts cache and memory of any size.**

The general section should be implemented by the whole team. On the other hand, the cache section should be divided by the team and each member should be responsible for the part chosen.

General

To be implemented by the whole team.

convertBinToDec/2

The predicate `convertBinToDec(Bin,Dec)` should succeed only if `Dec` is the decimal representation of the binary number `Bin` .

```
Example:
?- convertBinToDec(0101,D).
D = 5 ;
false.

?- convertBinToDec(1101,D).
D = 13 ;
```

```
false.

?- convertBinToDec(10,D).
D = 2 ;
false.
```

replaceIthItem/4

The predicate `replaceIthItem(Item,List,I,Result)` should succeed if `Result` is the result of replacing the item at position `I` in the list by the item `Item` .

```
Example:
?- replaceIthItem(a,[1,2,3,4],2,R).
R = [1, 2, a, 4] ;
false.

?- replaceIthItem(5,[1,2,3,4,9,8,7,6],5,R).
R = [1, 2, 3, 4, 9, 5, 7, 6] ;
false.
```

splitEvery/3

The predicate `splitEvery(N,List,Res)` succeeds when `Res` is a list in-which every `N` consecutive elements in the list `List` are grouped in a list maintaining the order

```
Example:
?- splitEvery(2,[a,b,c,d,e,f,g,h],R).
R = [[a, b], [c, d], [e, f], [g, h]] ;
false.

?- splitEvery(4,[a,b,c,d,e,f,g,h],R).
R = [[a, b, c, d], [e, f, g, h]] ;
false.

?- splitEvery(8,[a,b,c,d,e,f,g,h],R).
R = [[a, b, c, d, e, f, g, h]] ;
false.
```

logBase2/2

The predicate `logBase2(Num,Res)` succeeds when `Res` is equals to log_2(`Num`).

```
Example:
?- logBase2(8,N).
N = 3 ;
false.

?- logBase2(1,N).
N = 0 ;
false.

?- logBase2(32,N).
N = 5 ;
false.
```

getNumBits/4

Given the number of sets `NumOfSets`, the cache mapping type `Type` and the cache `Cache`, the predicate `getNumBits(NumOfSets,Type,Cache,BitsNum)` succeeds when `BitsNum` is required for the tag.

```
Examples:
?- getNumBits(_,fullyAssoc,[item(tag("00000"),data(1),0,0),
    item(tag("00000"),data(b),1,0),item(tag("00010"),data(0),0,3),
    item(tag("00000"),data(c),0,2)],BitsNum).

BitsNum = 0 ;
false.

?- getNumBits(2,setAssoc,[item(tag("00000"),data(1),0,0),
    item(tag("00000"),data(b),1,0),item(tag("00010"),data(0),0,3),
item(tag("00000"),data(c),0,2)],BitsNum).

BitsNum = 1 ;
false.

?- getNumBits(_,directMap,[item(tag("00000"),data(1),0,0),
    item(tag("00000"),data(b),1,0),item(tag("00010"),data(0),0,3),
item(tag("00000"),data(c),0,2)],BitsNum).

BitsNum = 2 ;
false.
```

fillZeros/4

Given a string representing a number `String` and a number `N`, the predicate `fillZeros(String,N,R)` succeeds when `R` is result of adding `N` preceding zeros to the string.

```
Examples:
?- fillZeros("100",1,R).
R = "0100" ;
false.

?- fillZeros("10",4,R).
R = "000010" ;
false.

?- fillZeros("0",2,R).
R = "000" ;
false.
```

Cache

To be divided by the team members

Cache Mapping 1: Direct Mapping

getDataFromCache/6

The predicate `getDataFromCache(StringAddress,Cache,Data,HopsNum,directMap,BitsNum)` should succeed when the `Data` is successfully retrieved from the `Cache` (cache hit) and the `HopsNum` represents the number of hops required to access the data from the cache which can differ according to direct map cache mapping technique such that:

- `StringAddress` is a string of the binary number which represents the address of the data you are required to address and it is six binary bits.
- `Cache` is the cache using the representation discussed previously .
- `Data` is the data retrieved from cache when cache hit occurs.
- `HopsNum` the number of hops required to access the data from the cache.
- `BitsNum` The BitsNum is the number of bits the index needs.

Example:

```
?- getDataFromCache("000001",[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(11000),1,0), item(tag("0001"),data(11100),0,3),
item(tag("0001"),data(11110),0,2)], Data, HopsNum, directMap,2).

Data = 11000,
HopsNum = 0 ;
false.


?- getDataFromCache("000010",
[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(11000),0,0), item(tag("0001"),data(11100),0,3),
item(tag("0001"),data(11110),0,2)],
Data, HopsNum, directMap,2).
false.


?- getDataFromCache("111101",
[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(11000),1,0), item(tag("0001"),data(11100),0,3),
item(tag("0001"),data(11110),0,2)],
Data, HopsNum, directMap,2).
false.
```

convertAddress/5

The predicate `convertAddress(Bin,BitsNum,Tag,Idx,directMap)` succeeds when `Tag` and `Idx` corresponds to the address`Bin` in the `directMap` cache mapping technique. The `BitsNum` is the number of bits the index needs.

Example:

```
?-  convertAddress(1110,2,Tag,Idx,directMap).
Tag = 11,
Idx = 10 ;
false.
```

```
?-  convertAddress(11100,2,Tag,Idx,directMap).
Tag = 111,
Idx = 0 ;
false.

?- convertAddress(000011,2,Tag,Idx,directMap).
Tag = 0,
Idx = 11 ;
false.
```

replaceInCache/8

The predicate`replaceInCache(Tag,Idx,Mem,OldCache,NewCache,ItemData, directMap,BitsNum)` should succeed when the data `ItemData` is successfully retrieved from the memory `Mem` and the cache `OldCache` is updated `NewCache` after replacing the item in cache using FIFO replace policy according to direct mapping cache mapping technique. The `BitsNum` is the number of bits the index needs.The tag is Tag and the index is Index are numbers in binary format. Note that: invalid data is considered as empty space and have the priority to replace over the order of replacement.

`Example:`

```
?- replaceInCache(0,10,["100000","100001","100010",
"100011","100100","100101","100110","100111"],
[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(100001),1,0),
item(tag("0001"),data(11100),0,3),
item(tag("0001"),data(11110),0,2)],
NewCache,Data,directMap,2).


NewCache = [item(tag("0000"), data(10000), 0, 1),
item(tag("0000"), data(100001), 1, 0),
item(tag("0000"), data("100010"), 1, 0),
item(tag("0001"), data(11110), 0, 2)],
Data = "100010" ;
false.
```

```
?- replaceInCache(0001,11,["100000","100001","100010",
"100011","100100","100101","100110","100111"],
[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(100001),1,0),
item(tag("0001"),data(11100),0,3),
item(tag("0000"),data(100011),1,2)],
NewCache,Data,directMap,2).

NewCache = [item(tag("0000"), data(10000), 0, 1),
item(tag("0000"), data(100001), 1, 0),
item(tag("0001"), data(11100), 0, 3),
item(tag("0001"), data("100111"), 1, 0)],
Data = "100111" ;
false.


?- replaceInCache(0001,11,
["100000","100001","100010","100011","100100",
"100101","100110","101011"],
[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(100001),1,0),
item(tag("0001"),data(11100),0,3),
item(tag("0001"),data(100011),0,2)],
NewCache,Data,directMap,2).

NewCache = [item(tag("0000"), data(10000), 0, 1),
item(tag("0000"), data(100001), 1, 0),
item(tag("0001"), data(11100), 0, 3),
item(tag("0001"), data("101011"), 1, 0)],
Data = "101011" ;
false.
```

getData/9

Note: This predicate is already implemented at the end of the description document. No need to re-implement it.

The predicate `getData(StringAddress,OldCache,Mem,NewCache,Data,HopsNum, directMap,BitsNum,Status)` should succeed when the `Data` stored in `StringAddress`

is retrieved from the `Cache` and the `HopsNum` represents the number of hops required to access the data from the cache in case of cache hit (number of tags compared) (number of tags compared) for direct mapping technique `directMap` and `Status` specifies whether it was cache hit or cache miss. `NewCache` will be the updated cache in case of a cache miss and will be the same in case of cache hit. `Mem` is the memory and `BitsNum` is the number of bits the index needs.

Example:

```
?- getData("000001",[item(tag("0000"),data(10000),0,1),
     item(tag("0000"),data(100001),1,0),
     item(tag("0001"),data(11100),0,3),
     item(tag("0001"),data(11110),0,2)],
     ["100000","100001","100010","100011","100100","100101","10011","100111"]
     ,NewCache,Data,HopsNum,directMap,2,Status).

NewCache = [item(tag("0000"), data(10000), 0, 1),
item(tag("0000"), data(100001), 1, 0),
item(tag("0001"), data(11100), 0, 3),
item(tag("0001"), data(11110), 0, 2)],
Data = 100001,
HopsNum = 0,
Status = hit ;
false.

getData("000001",[item(tag("0000"),data(10000),0,1),
     item(tag("0000"),data(100001),0,0),
     item(tag("0001"),data(11100),0,3),
     item(tag("0001"),data(11110),0,2)],
     ["100000","100101","100010","100011","100100","100101","10011","100111"]
     ,NewCache,Data,HopsNum,directMap,2,Status).

 NewCache = [item(tag("0000"), data(10000), 0, 1),
 item(tag("0000"), data("100101"), 1, 0),
 item(tag("0001"), data(11100), 0, 3),
 item(tag("0001"), data(11110), 0, 2)],
Data = "100101",
Status = miss ;
false.
```

runProgram/8

Note: This predicate is already implemented at the end of the description document. No need to re-implement it.

`runProgram(AdressList,OldCache,Mem,FinalCache,OutputDataList,StatusList, directMap,NumOfSets).` Given a list of addresses `AdressList`, `runProgram` predicate succeeds if `OutputDataList` is equals to the data retrieved from the cache `OldCache` in cache hit or Memory `Mem` in cache miss and the `FinalCache` is the updated cache after the retrieval of all the data. The `NumOfSets` is the same as the size of the cache. The runProgram should utilise direct mapping in this case.

```
Example:
?- runProgram(["000011","000100","000011","001000"],
[item(tag("0000"),data(10000),0,1),
item(tag("0000"),data(11000),0,0),
item(tag("0001"),data(11100),0,3),
item(tag("0001"),data(11110),0,2)],
[a,b,c,d,e,f,ab,ac,ad,ae,af],
FinalCache,OutputDataList,StatusList,directMap,4).

FinalCache = [item(tag("0010"), data(ad), 1, 0),
    item(tag("0000"), data(11000), 0, 0),
    item(tag("0001"), data(11100), 0, 3),
    item(tag("0000"), data(d), 1, 0)],
OutputDataList = [d, e, d, ad],
StatusList = [miss, miss, hit, miss] ;
false.
```

Cache Mapping 2: Fully Associative

getDataFromCache/6

The predicate `getDataFromCache(StringAddress,Cache,Data,HopsNum,fullyAssoc,BitsNum)` should succeed when the `Data` is successfully retrieved from the `Cache` (cache hit) and the `HopsNum` represents the number of hops required to access the data from the cache in fully-associative mapping technique such that:

- `StringAddress` is a string of the binary number which represents the address of the data you are required to address and it is six binary bits.
- `Cache` is the cache using the representation discussed previously .
- `Data` is the data retrieved from cache when cache hit occurs.

- `HopsNum` the number of hops required to access the data from the cache.
- `BitsNum` The BitsNum is the number of bits the index needs which will be zero in this case.

Example:

```
?- getDataFromCache("000001",
[item(tag("000000"),data(10000),0,1),
item(tag("000001"),data(11000),1,0),
item(tag("000100"),data(11100),0,3),
item(tag("000101"),data(11110),0,2)],
 Data, HopsNum, fullyAssoc,_).

Data = 11000,
HopsNum = 1 ;
false.

?- getDataFromCache("000011",
[item(tag("000000"),data(10000),1,1),
item(tag("000001"),data(11000),1,0),
item(tag("000100"),data(11100),1,3),
item(tag("000101"),data(11110),1,2)],
Data, HopsNum, fullyAssoc,_).
false.



 getDataFromCache("000011",
 [item(tag("000011"),data(10000),0,1),
 item(tag("000001"),data(11000),1,0),
 item(tag("000100"),data(11100),1,3),
 item(tag("000101"),data(11110),1,2)],
 Data, HopsNum, fullyAssoc,_).
false.
```

convertAddress/5

The predicate `convertAddress(Bin,BitsNum,Tag,Idx,fullyAssoc)` succeeds when `Tag` and `Idx` corresponds to the address`Bin` in the `fullyAssoc` cache mapping tech-

nique. The `BitsNum` is the number of bits the index needs which will be zero in this case.

`Example:`

```
?- convertAddress(001110,BitsNum,Tag,Idx,fullyAssoc).
Tag = 1110 ;
false.

?- convertAddress(000011,BitsNum,Tag,Idx,fullyAssoc).
Tag = 11 ;
false.

?- convertAddress(000001,BitsNum,Tag,Idx,fullyAssoc).
Tag = 1 ;
false.
```

replaceInCache/8

The predicate `replaceInCache(Tag,Idx,Mem,OldCache,NewCache,ItemData,fullyAssoc, BitsNum)` should succeed when the data `ItemData` is successfully retrieved from the memory `Mem` and the cache `OldCache` is updated `NewCache` after replacing the item in cache using FIFO replace policy according to Fully-associative cache mapping technique. The `BitsNum` is the number of bits the index needs.The tag is Tag and the index is Index are numbers in binary format. Note that: invalid data is considered as empty space and have the priority to replace over the order of replacement.

`Example:`

```
?- replaceInCache(1,0,
["100000","100001","100010","100011","100100","100101","100110","100111"],
[item(tag("000000"),data(10000),0,1),
item(tag("000010"),data("100010"),1,0),
item(tag("000100"),data(11100),0,3),
item(tag("000101"),data(11110),0,2)],
NewCache,ItemData,fullyAssoc,_).

NewCache = [item(tag("000001"), data("100001"), 1, 0),
item(tag("000010"), data("100010"), 1, 1),
```

```
item(tag("000100"), data(11100), 0, 3),
item(tag("000101"), data(11110), 0, 2)],
ItemData = "100001" ;
false.

?- replaceInCache(1,0,
["100000","100001","100010","100011","100100","100101","100110","100111"],
[item(tag("000000"),data(10000),1,1),
item(tag("000010"),data(100100),1,0),
item(tag("000100"),data(100100),1,3),
item(tag("000101"),data(100101),1,2)],
NewCache,ItemData,fullyAssoc,_).

NewCache = [item(tag("000000"), data(10000), 1, 2),
item(tag("000010"), data(100100), 1, 1),
item(tag("000001"), data("100001"), 1, 0),
item(tag("000101"), data(100101), 1, 3)],
ItemData = "100001" ;
false.

?- replaceInCache(1,0,
["100000","100001","100010","100011","100100","100101","100110","100111"],
[item(tag("000001"),data(100001),0,1),
item(tag("000010"),data(100010),1,0),
item(tag("000100"),data(100100),1,3),
item(tag("000101"),data(100101),1,2)],
NewCache,ItemData,fullyAssoc,_).

NewCache = [item(tag("000001"), data("100001"), 1, 0),
item(tag("000010"), data(100010), 1, 1),
item(tag("000100"), data(100100), 1, 4),
item(tag("000101"), data(100101), 1, 3)],
ItemData = "100001" ;
false.
```

getData/9

Note: This predicate is already implemented at the end of the description document. No need to re-implement it.
The predicate The predicate `getData(StringAddress,OldCache,Mem,NewCache,Data,HopsNum,`

`fullyAssoc,BitsNum,Status)` should succeed when the `Data` stored in `StringAddress` is retrieved from the `Cache` and the `HopsNum` represents the number of hops required to access the data from the cache in case of cache hit (number of tags compared) in fully-associative mapping technique `fullyAssoc` used and `Status` specifies whether it was cache hit or cache miss. `NewCache` will be the updated cache in case of a cache miss and will be the same in case of cache hit. `Mem` is the memory and `BitsNum` is the number of bits the index needs which will be zero in this case.

`Examples:`

```
?- getData("000001",
[item(tag("000000"),data(10000),1,1), item(tag("000001"),data(11000),1,0),
item(tag("00010"),data(11100),0,3),item(tag("00000"),data(11110),0,2)],
["100000","100001","100010","100011","100100","100101","100110","100111"],
NewCache,Data,HopsNum,fullyAssoc,_,Status).

NewCache = [item(tag("000000"), data(10000), 1, 1),
    item(tag("000001"), data(11000), 1, 0),
    item(tag("00010"), data(11100), 0, 3),
    item(tag("00000"), data(11110), 0, 2)],
Data = 11000,
HopsNum = 1,
Status = hit ;
false.


?- getData("000001",
[item(tag("000000"),data(10000),1,1),
item(tag("000001"),data(11000),1,0),
item(tag("00010"),data(11100),0,3),
item(tag("00000"),data(11110),0,2)],
["100000","100001","100010","100011","100100","100101","100110","100111"]
,NewCache,Data,HopsNum,fullyAssoc,_,Status).

NewCache = [item(tag("000000"), data(10000), 1, 1),
item(tag("000001"), data(11000), 1, 0),
item(tag("00010"), data(11100), 0, 3),
item(tag("00000"), data(11110), 0, 2)],
Data = 11000,
HopsNum = 1,
```

```
Status = hit ;
false.
```

```
?-getData("000001",
[item(tag("000001"),data(100001),0,1),
item(tag("000010"),data(100010),1,0),
item(tag("000100"),data(100100),1,3),
item(tag("000101"),data(100101),1,2)],
["100000","100001","100010","100011","100100","100101","100110","100111"],
NewCache,Data,HopsNum,fullyAssoc,_,Status).

NewCache = [item(tag("000001"), data("100001"), 1, 0),
item(tag("000010"), data(100010), 1, 1),
item(tag("000100"), data(100100), 1, 4),
item(tag("000101"), data(100101), 1, 3)],
Data = "100001",
Status = miss ;
false.
```

runProgram/8

Note: This predicate is already implemented at the end of the description document. No need to re-implement it.

`runProgram(AdressList,OldCache,Mem,FinalCache,OutputDataList,StatusList, fullyAssoc,NumOfSets)`. Given a list of addresses `AdressList`, `runProgram` predicate succeeds if `OutputDataList` is equals to the data retrieved from the cache `OldCache` in cache hit or Memory `Mem` in cache miss and the `FinalCache` is the updated cache after the retrieval of all the data. The `NumOfSets` will always be one since it is considered as one big set. The runProgram should utilise fully-associative in this case.

```
?- runProgram(["000000","000001","000000","000011"],
    [item(tag("0000"),data(10000),0,0), item(tag("0000"),data(11000),0,0),
    item(tag("0001"),data(11100),0,3),  item(tag("00001"),data(e),0,0)],
    [a,b,c,d,e,f,ab,ac,ad,ae],FinalCache,OutputDataList,StatusList,fullyAssoc,1).

FinalCache = [item(tag("000000"), data(a), 1, 2),
item(tag("000001"), data(b), 1, 1),
```

```
item(tag("000011"), data(d), 1, 0),
item(tag("00001"), data(e), 0, 0)],
OutputDataList = [a, b, a, d],
StatusList = [miss, miss, hit, miss] ;
false.
```

Cache Mapping 3: Set-associative

getDataFromCache/6

The predicate `getDataFromCache(StringAddress,Cache,Data,HopsNum,setAssoc,SetsNum)` should succeed when the `Data` is successfully retrieved from the `Cache` (cache hit) and the `HopsNum` represents the number of hops required to access the data from the cache in set-associative mapping technique such that:

- `StringAddress` is a string of the binary number which represents the address of the data you are required to address and it is six binary bits.
- `Cache` is the cache using the representation discussed previously .
- `Data` is the data retrieved from cache when cache hit occurs.
- `HopsNum` the number of hops required to access the data from the cache.
- `SetsNum` is the number of sets

Examples:

```
Hit case
?- getDataFromCache("000001",[item(tag("00000"), data(11100), 0, 3),
item(tag("00000"), data(11110), 0, 2),
item(tag("0000"), data(10000), 0, 1),
item(tag("00000"), data(11000), 1, 0)],
Data, HopsNum, setAssoc,1).

Data = 11000,
HopsNum = 1 ;
false.

Miss case
?- getDataFromCache("000001",
[item(tag("00000"),data(10000),0,1), item(tag("00000"),data("100000"),1,0),
item(tag("00010"),data(11100),0,3),item(tag("00000"),data("11110"),0,2)],
```

```
Data,HopsNum,setAssoc,1).

false.


Miss case
getDataFromCache("000000",
[item(tag("00000"),data("10000"),0,1), item(tag("00001"),data("11000"),1,0),
item(tag("00010"),data("11100"),0,3),item(tag("00000"),data("11110"),1,0)],
Data,HopsNum,setAssoc,1).

false.


Hit case
getDataFromCache("000000",
[item(tag("00000"),data("10000"),1,1), item(tag("00001"),data("11000"),1,0),
item(tag("00010"),data("11100"),0,3),item(tag("00000"),data("11110"),1,0)],
Data,HopsNum,setAssoc,1).

Data = "10000",
HopsNum = 0 ;
false.
```

convertAddress/5

The predicate `convertAddress(Bin,SetsNum,Tag,Idx,setAssoc)` succeeds when `Tag` and `Idx` corresponds to the address`Bin` in the `setAssoc` cache mapping technique. The `SetsNum` is the number of sets.

```
Example:

?- convertAddress(001110,4,Tag,Idx,setAssoc).
Tag = 11,
Idx = 10.

?- convertAddress(000011,4,Tag,Idx,setAssoc).
Tag = 0,
```

```
Idx = 11.

?- convertAddress(001110,8,Tag,Idx,setAssoc).
Tag = 1,
Idx = 110.

?- convertAddress(001110,1,Tag,Idx,setAssoc).
Tag = 1110,
Idx = 0.
```

replaceInCache/8

The predicate `replaceInCache(Tag,Idx,Mem,OldCache,NewCache,ItemData, setAssoc,SetsNum)` should succeed when the data `ItemData` is successfully retrieved from the memory `Mem` and the cache `OldCache` is updated `NewCache` after replacing the item in cache using FIFO replace policy according to set-associative cache mapping technique. The tag is `Tag` and the index is `Index` are numbers in binary format. The `SetsNum` is the number of sets. Note that: invalid data is considered as empty space and have the priority to replace over the order of replacement.

Examples:

```
?- replaceInCache(0,1,
["100000","100001","100010","100011","100100","100101","100110","100111"],
[item(tag("00000"),data(10000),0,1), item(tag("00000"),data("100000"),1,0),
item(tag("00010"),data(11100),0,3),item(tag("00000"),data("11110"),0,2)],
NewCache,Data,setAssoc,2).

NewCache = [item(tag("00000"), data(10000), 0, 1),
item(tag("00000"), data("100000"), 1, 0),
item(tag("00000"), data("100001"), 1, 0),
item(tag("00000"), data("11110"), 0, 2)],
Data = "100001" ;
false.

?- replaceInCache(11,1,
["100000","100001","100010","100011","100100","100101","100110","100111"],
[item(tag("00000"),data(10000),0,1), item(tag("00000"),data("100000"),1,0),
item(tag("00001"),data("100011"),1,0),item(tag("00000"),data("100001"),1,1)],
```

```
NewCache,Data,setAssoc,2).

NewCache = [item(tag("00000"), data(10000), 0, 1),
item(tag("00000"), data("100000"), 1, 0),
item(tag("00001"), data("100011"), 1, 1),
item(tag("00011"), data("100111"), 1, 0)],
Data = "100111" ;
false.


replaceInCache(0,0,
["100000","100001","100010","100011","100100","100101","100110","100111"],
[item(tag("00000"),data(10000),0,0), item(tag("00000"),data("100000"),1,0),
item(tag("00010"),data(11100),0,3),item(tag("00000"),data("11110"),0,2)],
NewCache,Data,setAssoc,2).

NewCache = [item(tag("00000"), data("100000"), 1, 0),
item(tag("00000"), data("100000"), 1, 1),
item(tag("00010"), data(11100), 0, 3),
item(tag("00000"), data("11110"), 0, 2)],
Data = "100000" ;
false.
```

getData/9

Note: This predicate is already implemented at the end of the description document. No need to re-implement it.

The predicate `getData(StringAddress,OldCache,Mem,NewCache,Data,HopsNum, setAssoc,NumOfSets,Status)` should succeed when the `Data` stored in `StringAddress` is retrieved from the `Cache` and the `HopsNum` represents the number of hops required to access the data from the cache in case of cache hit (number of tags compared) in set-associative mapping technique `setAssoc` used and `Status` specifies whether it was cache hit or cache miss. `NewCache` will be the updated cache in case of a cache miss and will be the same in case of cache hit. `Mem` is the memory and `NumOfSets` is the number of sets.

```
Examples:
1 ?- getData("000001",
[item(tag("00000"),data(10000),0,1), item(tag("00000"),data("100000"),1,0),
item(tag("00010"),data(11100),0,3),item(tag("00000"),data("11110"),0,2)],
["100000","100001","100010","100011","100100","100101","100110","100111"],
```

```
NewCache,Data,HopsNum,setAssoc,2,Status).

NewCache = [item(tag("00000"), data(10000), 0, 1),
    item(tag("00000"), data("100000"), 1, 0),
    item(tag("00000"), data("100001"), 1, 0),
    item(tag("00000"), data("11110"), 0, 2)],
Data = "100001",
Status = miss ;
false.


2 ?- getData("000001",
[item(tag("00000"),data("10000"),0,1), item(tag("00000"),data("11000"),1,0),
item(tag("00010"),data("11100"),0,3),item(tag("00000"),data("11110"),1,0)],
["11000","11110","100010","100011","100100","100101","100110","100111"],
NewCache,Data,HopsNum,setAssoc,2,Status).

NewCache = [item(tag("00000"), data("10000"), 0, 1),
    item(tag("00000"), data("11000"), 1, 0),
    item(tag("00010"), data("11100"), 0, 3),
    item(tag("00000"), data("11110"), 1, 0)],
Data = "11110",
HopsNum = 1,
Status = hit ;
false.

3 ?- getData("000000",
[item(tag("00000"),data("10000"),0,1), item(tag("00001"),data("11000"),1,0),
item(tag("00010"),data("11100"),0,3),item(tag("00000"),data("11110"),1,0)],
["100000","11110","11000","100011","100100","100101","100110","100111"],
NewCache,Data,HopsNum,setAssoc,2,Status).

NewCache = [item(tag("00000"), data("100000"), 1, 0),
    item(tag("00001"), data("11000"), 1, 1),
    item(tag("00010"), data("11100"), 0, 3),
    item(tag("00000"), data("11110"), 1, 0)],
Data = "100000",
Status = miss ;
false.
```

German University in Cairo
Faculty of Media Engineering and Technology
Dr. Nada Sharaf

runProgram/8

Note: This predicate is already implemented at the end of the description document. No need to re-implement it.
`runProgram(AdressList,OldCache,Mem,FinalCache,OutputDataList,StatusList, setAssoc,NumOfSets).` Given a list of addresses `AdressList`, `runProgram` predicate succeeds if `OutputDataList` is equals to the data retrieved from the cache `OldCache` in cache hit or Memory `Mem` in cache miss and the `FinalCache` is the updated cache after the retrieval of all the data. `NumOfSets` is the number of sets in the cache. The runProgram should utilise set-associative in this case.

```
?- runProgram(["000011","000100","000011","001100"],
    [item(tag("0000"),data(10000),0,0), item(tag("0000"),data(11000),0,0),
    item(tag("0001"),data(11100),0,3), item(tag("00001"),data(e),0,0)],
    [a,b,c,d,e,f,ab,ac,ad,ae,af,a,a,a,a,aa,a],
    FinalCache,OutputDataList,StatusList,setAssoc,2).

FinalCache = [item(tag("00010"), data(e), 1, 1),
    item(tag("00110"), data(a), 1, 0),
    item(tag("00001"), data(d), 1, 0),
    item(tag("00001"), data(e), 0, 0)],
OutputDataList = [d, e, d, a],
StatusList = [miss, miss, hit, miss] ;
false.
```

Implemented Predicates

getData/9

```
getData(StringAddress,OldCache,Mem,NewCache,Data,HopsNum,Type,BitsNum,hit):-
getDataFromCache(StringAddress,OldCache,Data,HopsNum,Type,BitsNum),
NewCache = OldCache.

getData(StringAddress,OldCache,Mem,NewCache,Data,HopsNum,Type,BitsNum,miss):-
\+getDataFromCache(StringAddress,OldCache,Data,HopsNum,Type,BitsNum),
atom_number(StringAddress,Address),
convertAddress(Address,BitsNum,Tag,Idx,Type),
replaceInCache(Tag,Idx,Mem,OldCache,NewCache,Data,Type,BitsNum).
```

German University in Cairo
Faculty of Media Engineering and Technology
Dr. Nada Sharaf

runProgram/8

```
runProgram([],OldCache,_,OldCache,[],[],Type,_).
runProgram([Address|AdressList],OldCache,Mem,FinalCache,
    [Data|OutputDataList],[Status|StatusList],Type,NumOfSets):-
getNumBits(NumOfSets,Type,OldCache,BitsNum),
getData(Address,OldCache,Mem,NewCache,Data,HopsNum,Type,BitsNum,Status),
runProgram(AdressList,NewCache,Mem,FinalCache,OutputDataList,StatusList,
    Type,NumOfSets).
```