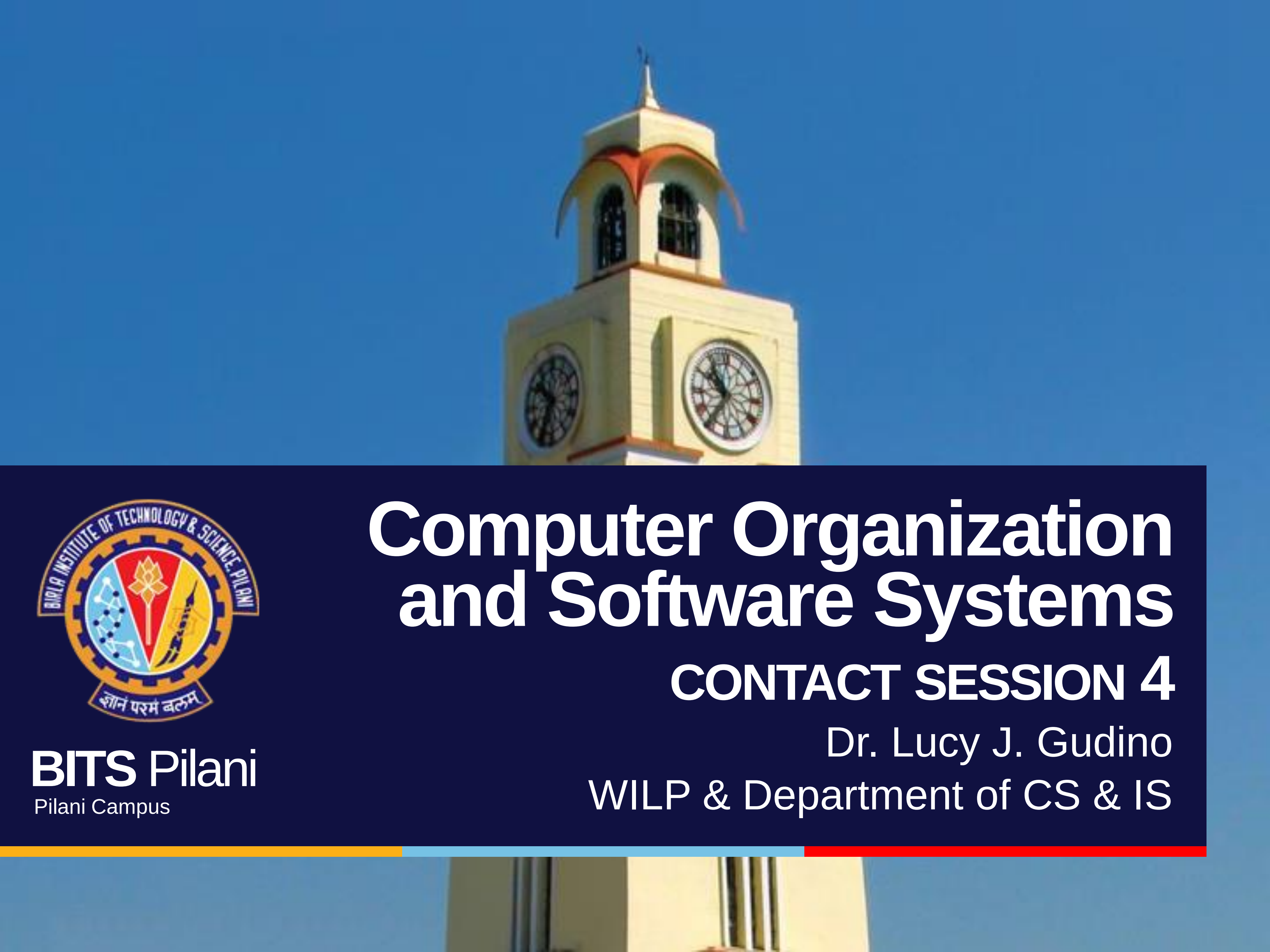


BITS Pilani
Pilani Campus

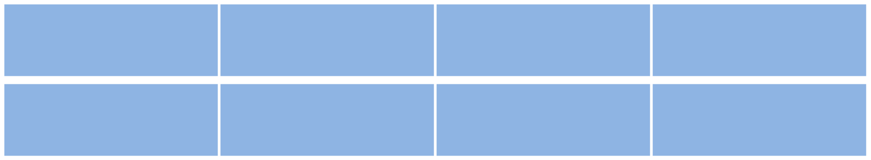
Computer Organization and Software Systems

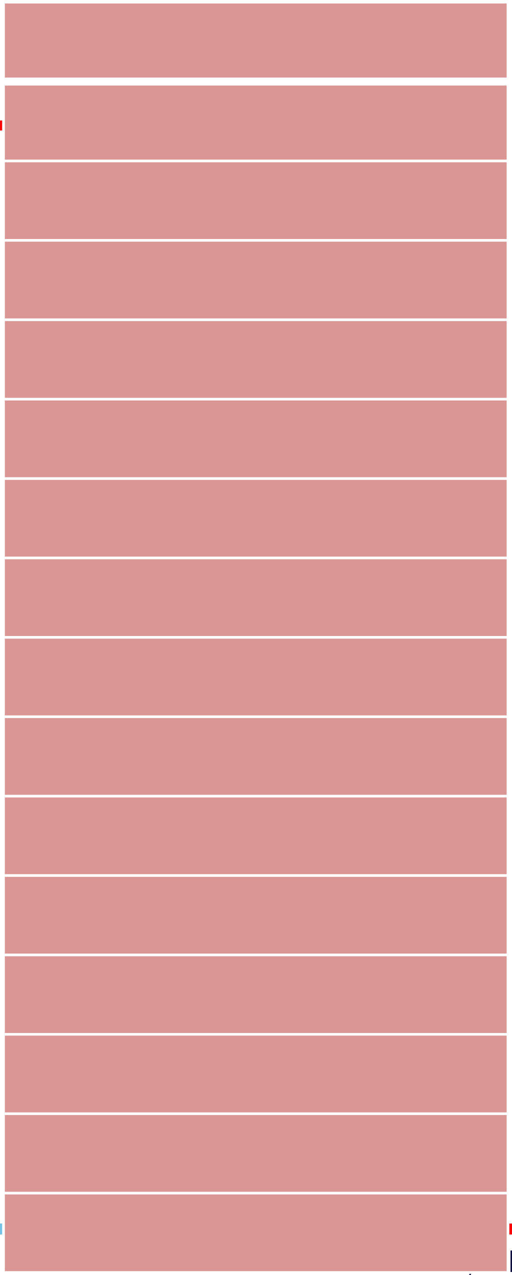
CONTACT SESSION 4

Dr. Lucy J. Gudino
WILP & Department of CS & IS

Direct Mapped Cache

- 16 Bytes main memory
  - How many address bits are required?
- Memory block size is 4 bytes
- Cache of 8 Byte
  - How many cache lines?
  - cache contains 2 lines (4 bytes per Line)

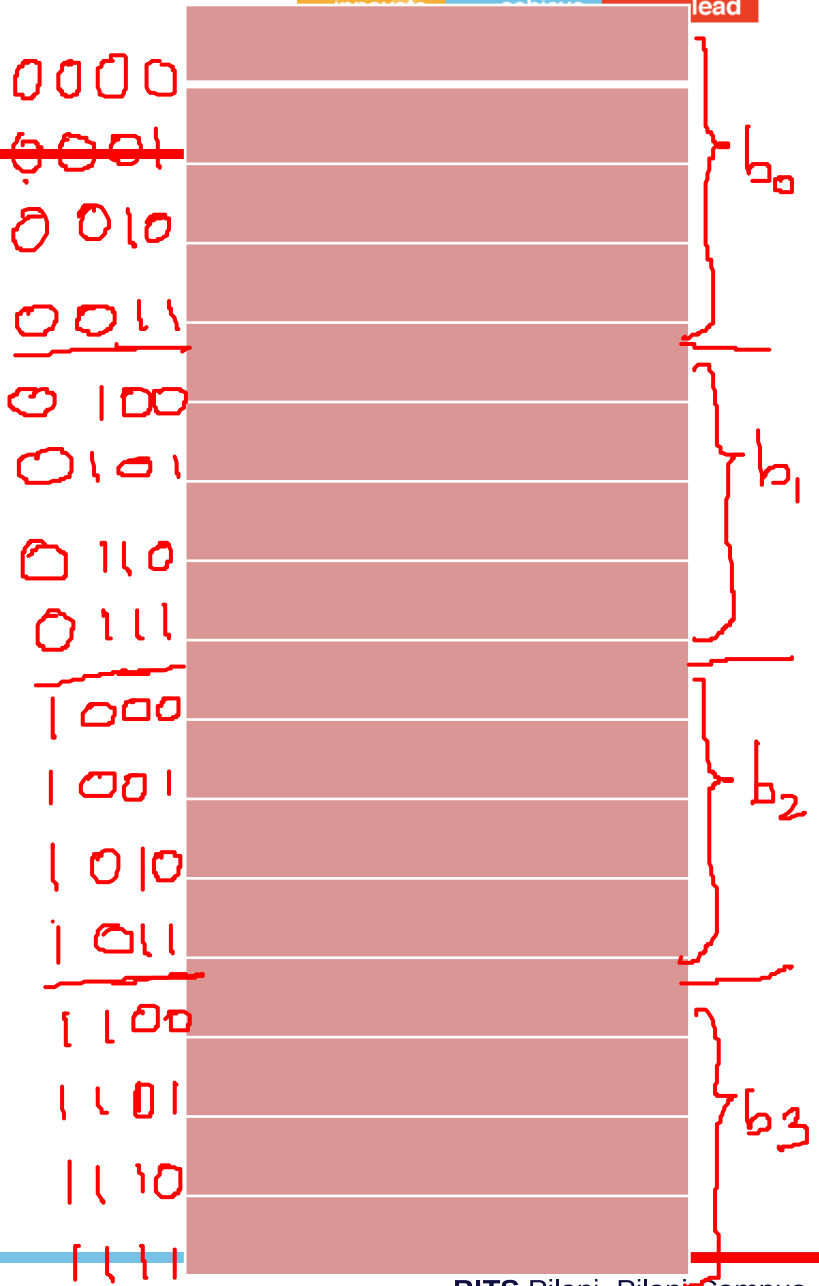




Direct Mapped Cache

i = j modulo m

- Address is split in three parts:
  - Tag
  - Line
  - Word



Direct mapped cache- Summary

1	1	2
Tag s-r	Line or Slot r	Word w

Direct mapped cache-pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Problem 1 : Direct Mapped Cache

Given :

- Cache of 64KByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

a) Number of bits required to address the main memory

b) Number of blocks in main memory

c) Number of cache lines

d) Number of bits required to identify a word (byte) in a block

e) Number of bits to identify a block

f) Tag, Line, Word

Solution 1

Given :

- Cache of 64kByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

a) Number of bits required to address the main memory

b) Number of blocks in main memory

c) Number of cache lines

Solution 1

Given :

- Cache of 64kByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

d) Number of bits required to identify a word (byte) in a block?

e) Number of bits required to identify a block

f) Tag, Line, Word

Tag s-r	Line r	Word w
8	14	2

Problem 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a. How is a 16-bit memory address divided into tag, line number, and byte number?

b. Into what line would bytes with each of the following addresses be stored?

```
0001   0001   0001   1011
1100   0011   0011   0100
1101   0000   0001   1101
1010   1010   1010   1010
```

c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

d. How many total bytes of memory can be stored in the cache?

e. Why is the tag also stored in the cache?

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a. How is a 16-bit memory address divided into tag, line number, and byte number?

b. Into what line would bytes with each of the following addresses be stored?

```
0001   0001   0001   1011
1100   0011   0011   0100
1101   0000   0001   1101
1010   1010   1010   1010
```

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

0001 1010 0001 1**000**
0001 1010 0001 1**001**
0001 1010 0001 1**010** **: given in the problem**
0001 1010 0001 1**011**
0001 1010 0001 1**100**
0001 1010 0001 1**101**
0001 1010 0001 1**110**
0001 1010 0001 1**111**

Solution 2

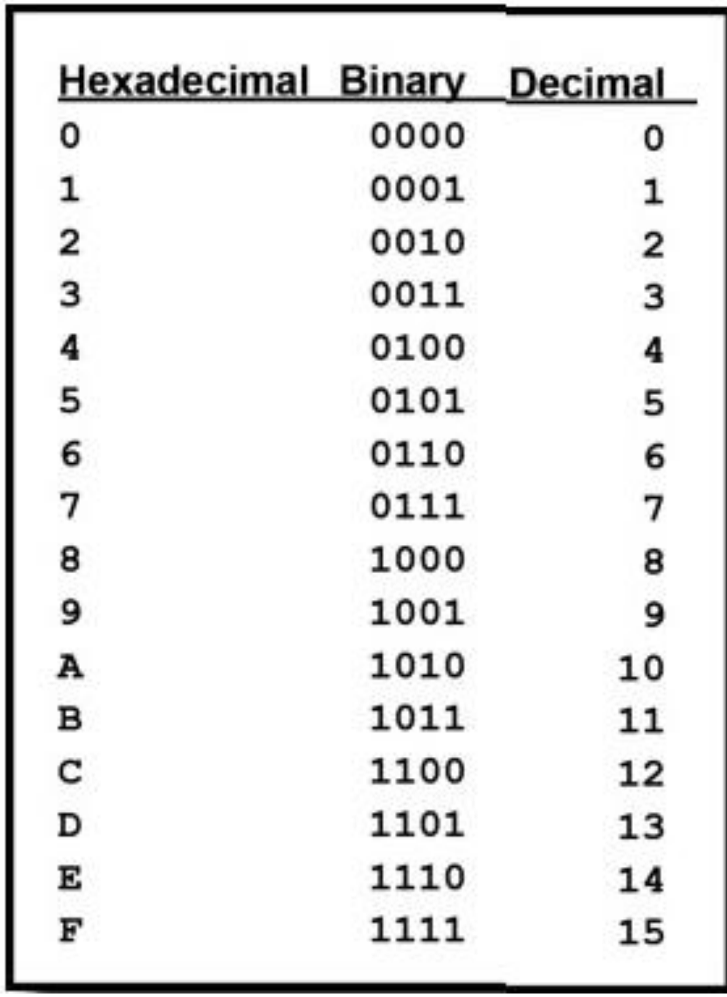
Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

d. How many total bytes of memory can be stored in the cache?

e. Why is the tag also stored in the cache?

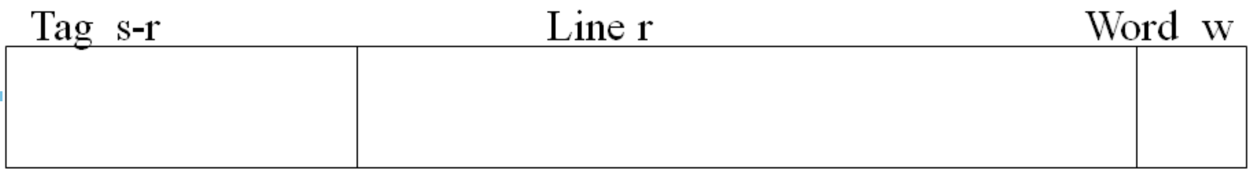
Problem 3

Consider a direct-mapped cache with 64 cache lines and a block size of 16 bytes and main memory of 8K (Byte addressable memory) . To what line number does byte address 1200H map?



Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Problem 4



The system uses a L1 cache with direct mapping and 32-bit address format is as follows:

bits 0 - 3 = offset (word)

bits 4 - 14 = index bits (Line)

bits15 - 31 = tag

a) What is the size of cache line?

b) How many Cache lines are there?

c) How much space is required to store the tags in the L1 cache?

d) What is the total Capacity of cache including tag storage?

Problem 5

- 16 Bytes main memory,
  Memory block size is 4 bytes,
  Cache of 8 Byte (cache is 2 lines of 4 bytes each)
- Block access sequence :

0 2 0 2 2 0 0 2 0 0 0 2 1

- Find out hit ratio.

Problem 6

Suppose a 1024-byte cache has an access time of 0.1 microseconds and the main memory stores 1 Mbytes with an access time of 1 microsecond. A referenced memory block that is not in cache must be loaded into cache .

Answer the following questions:

a) What is the number of bits needed to address the main memory?

b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

Solution 6

a) 20 bits

b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

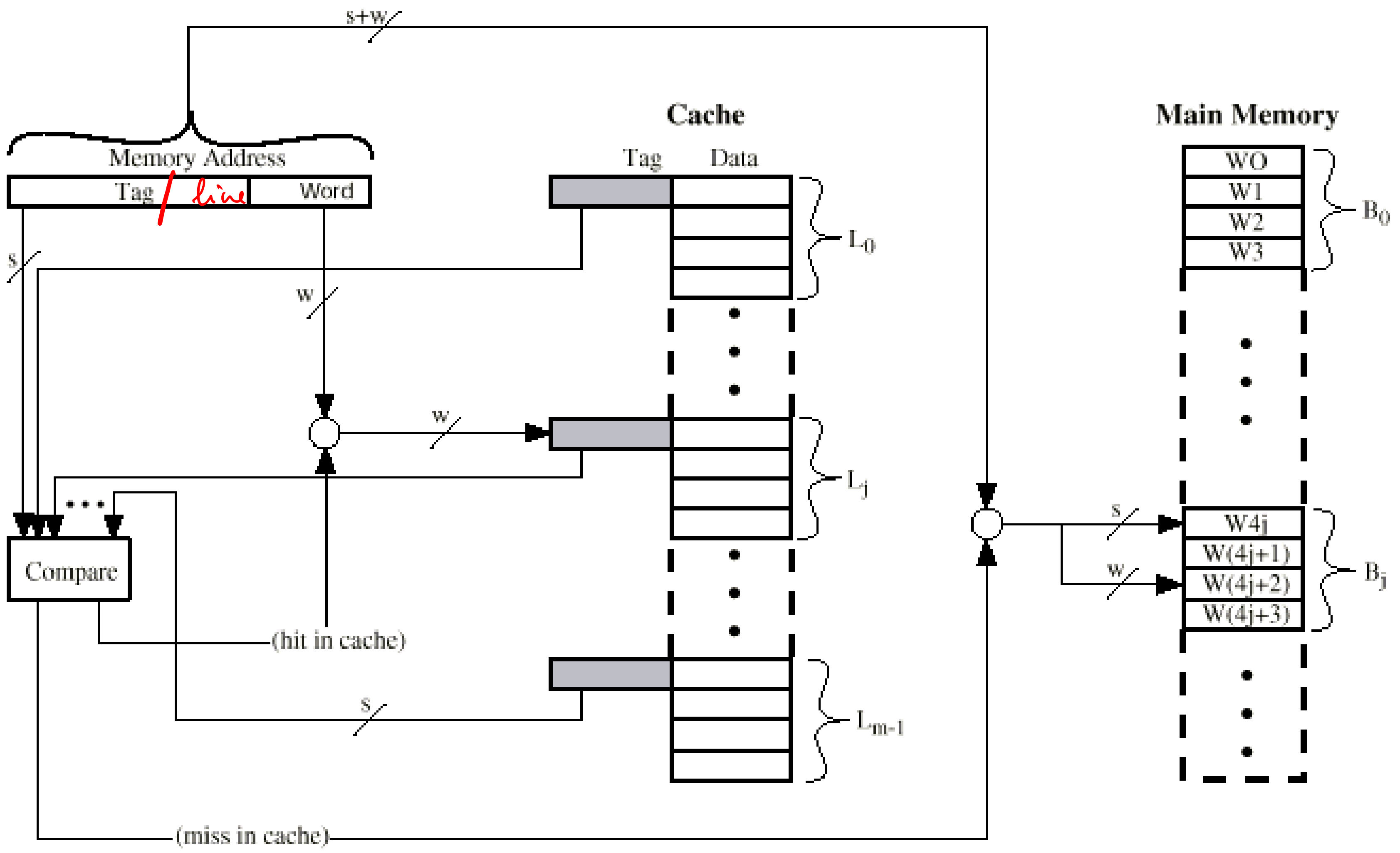
Avg access time = hit ratio * cache access +

(1- hit ratio) * (cache access + memory access)

Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

Associative Cache Organization



Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory
b) Number of blocks in main memory
c) Number of cache lines
d) Number of bits required to identify a word (byte) in a block?
e) Tag, Word

Problem 8

Cache of 64KByte, Cache block of 4 bytes and 16 M Bytes main memory and associative mapping.

Fill in the blanks:

Number of bits in main memory address = ______

Number of lines in the cache memory = _______

Word bits = __________

Tag bits = _________

Problem 9

- 16 Bytes main memory, Memory block size is 4 bytes, Cache of 8 Byte (cache is 2 lines of 4 bytes each) and associative mapping
- Block access sequence :

  0 2 0 2 2 0 0 2 0 0 0 2 1

- Find out hit ratio.

Set Associative Mapping

- Cache is divided into a number of sets (v sets each with k lines)
- m = v * k

  i = j modulo v

  where i = cache set number

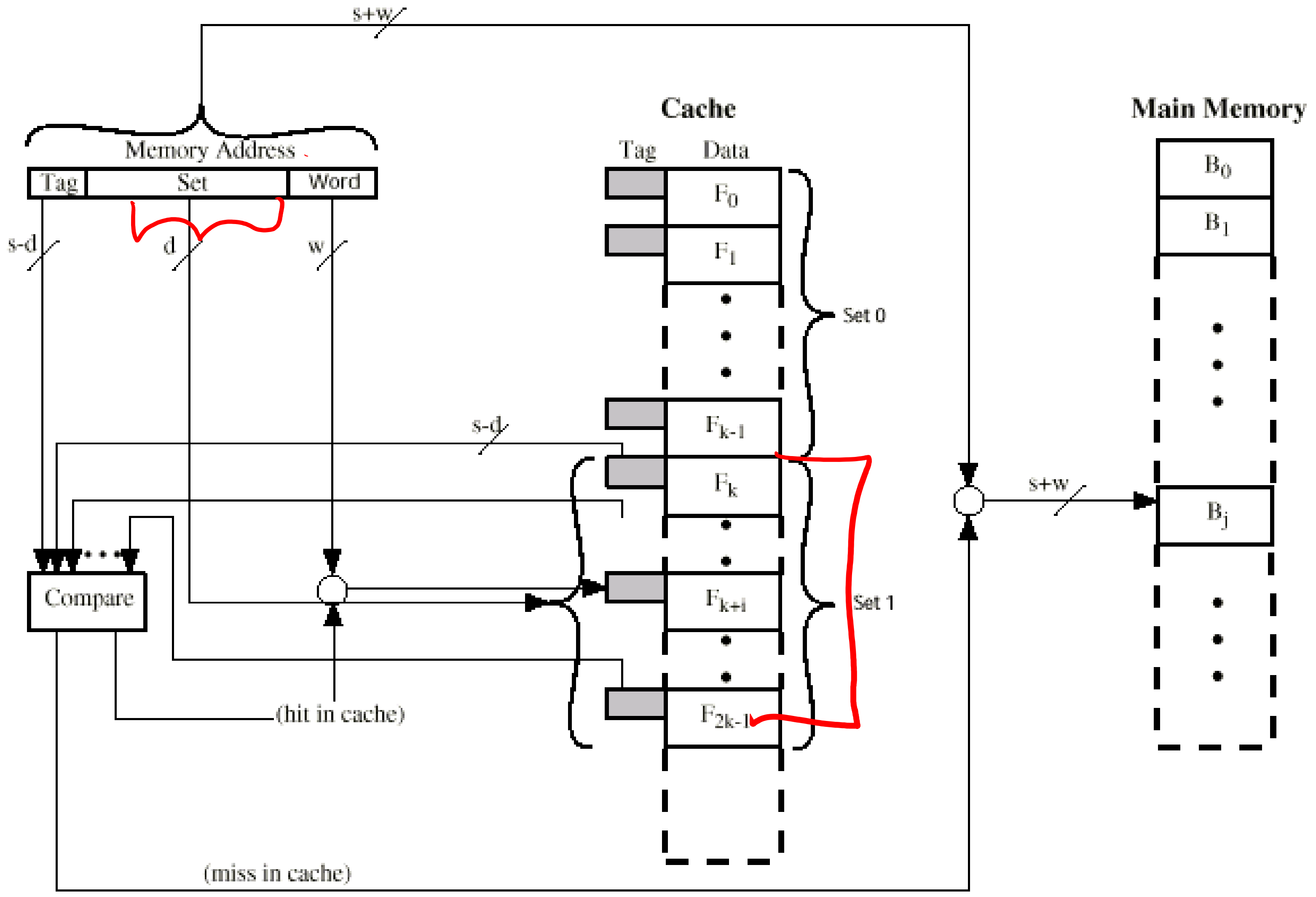
  j = main memory block number

  m = number of lines in the cache

Tag	Set	Word

- Each set contains 'k' number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way set associative mapping
  - A given block can be in one of 2 lines in only one set

Two Way Set Associative Cache Organization



Set Associative Mapping Summary

Address length = $(s + w)$ bits

Number of addressable units = 2^{s+w} words or bytes

Block size = line size = 2^w words or bytes

Number of blocks in main memory = 2^d

Number of lines in set = k

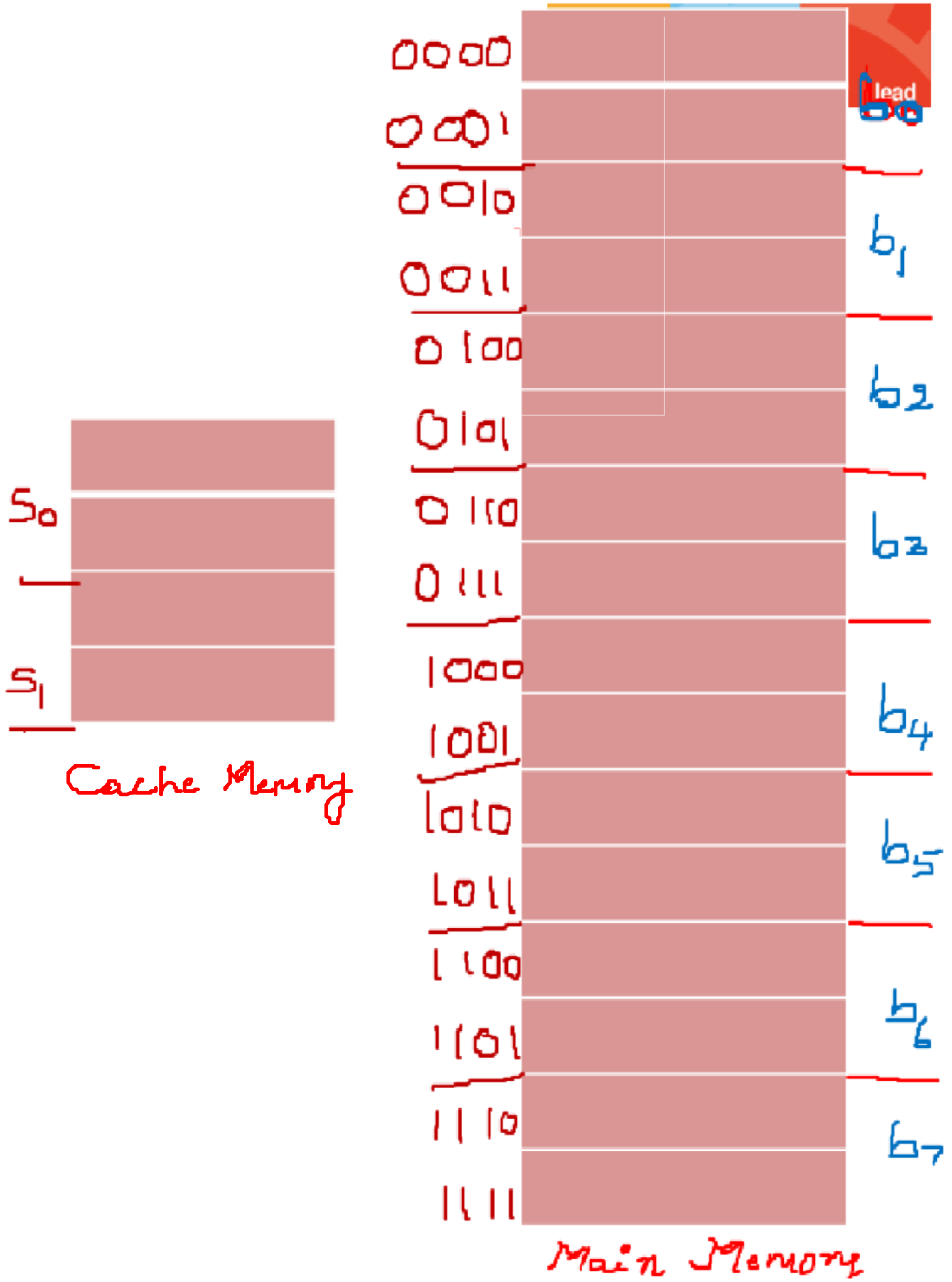
Number of sets = $v = 2^d$

Number of lines in cache = $kv = k * 2^d$

Size of tag = $(s - d)$ bits

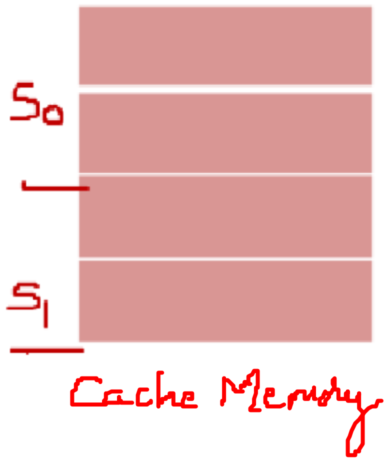
Example

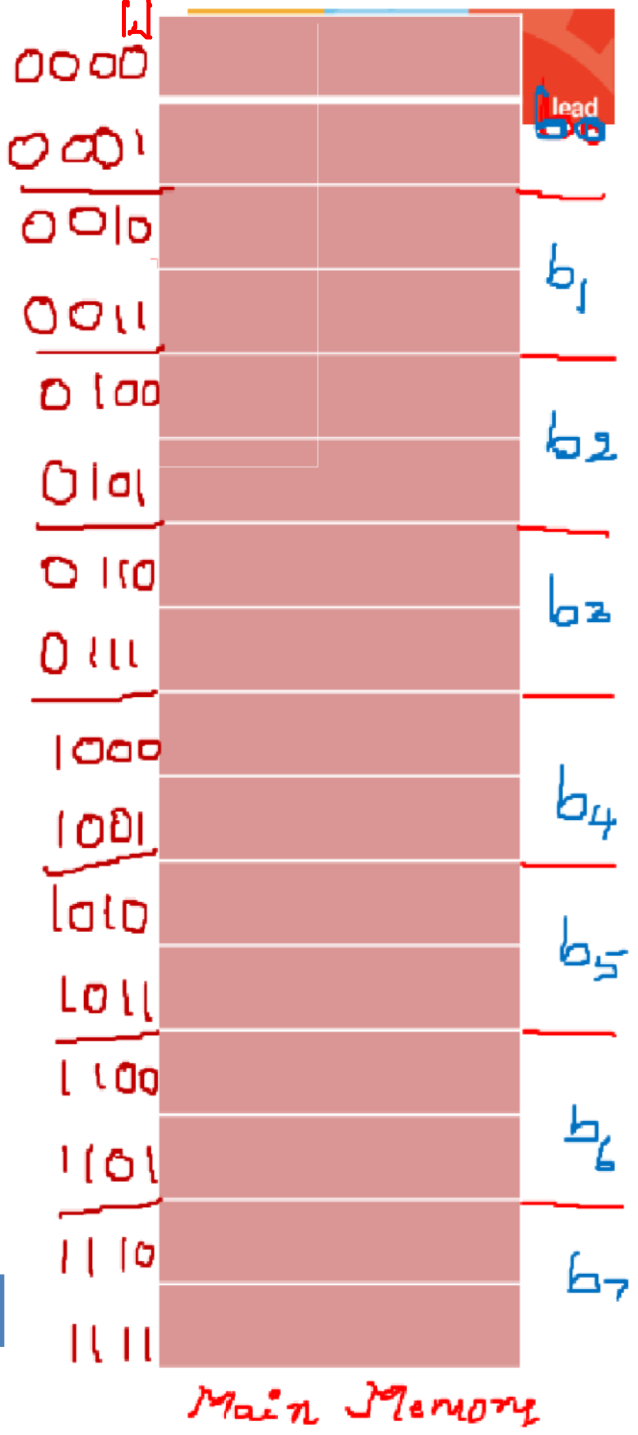
- 16 Bytes main memory, Block Size is 2 Bytes,
- Cache of 8 Bytes, 2 way set associative cache
  - # address bits
  - Cache line size
  - # main memory blocks
  - # Number of cache lines
  - # lines per set
  - # of sets



Example – Mapping Function

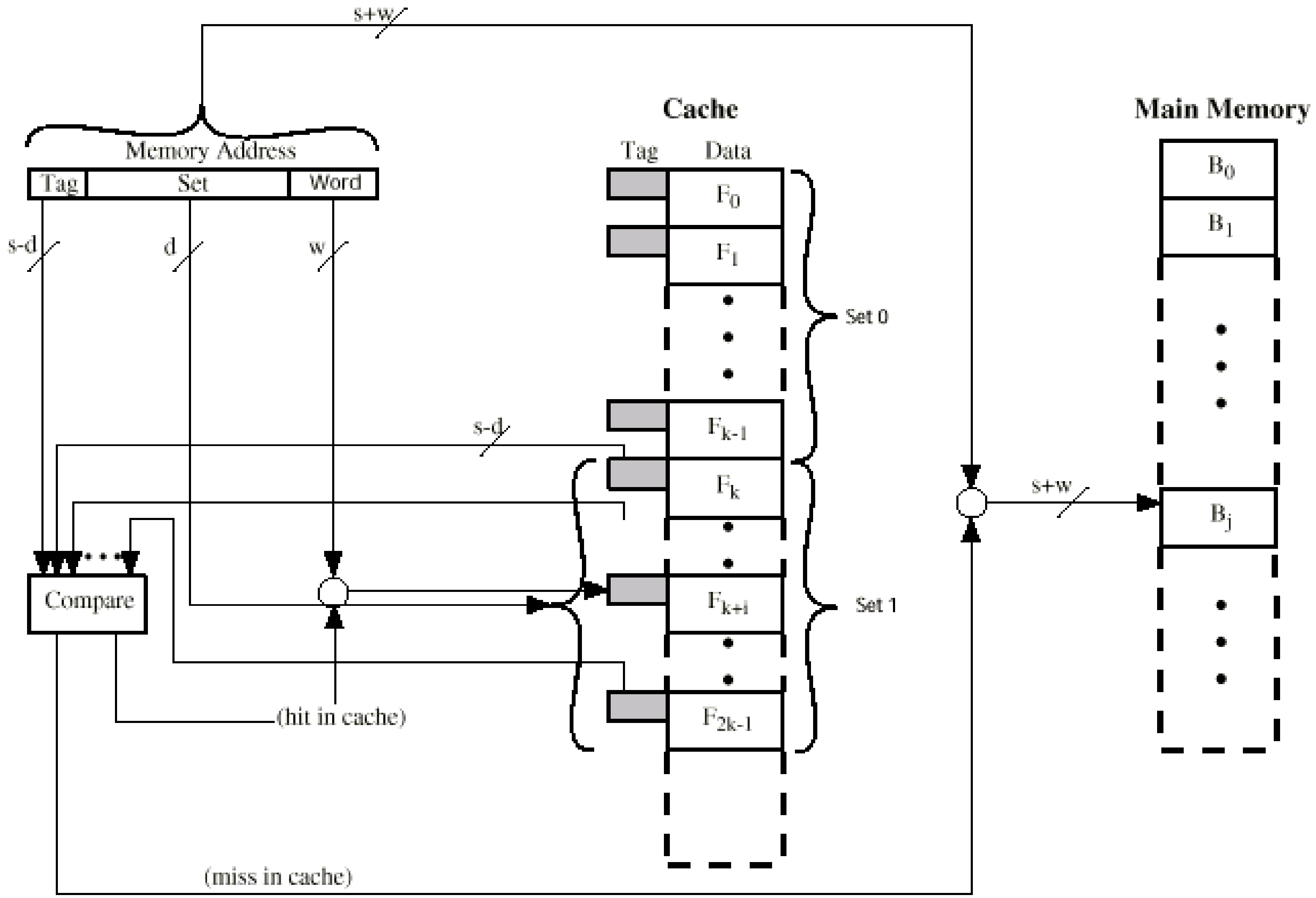
i = j modulo v	Set #
0%2	
1%2	
2%2	
3%2	
4%2	
5%2	
6%2	
7%2	





TAG	SET	WORD

Set Associative Cache Organization



Set Associative Mapping Summary

Address length = $(s + w)$ bits

Number of addressable units = 2^{s+w} words or bytes

Block size = line size = 2^w words or bytes

Number of blocks in main memory = 2^s

Number of lines in set = k

Number of sets = $v = 2^d$

Number of lines in cache = $kv = k * 2^d$

Size of tag = $(s - d)$ bits

Problem 1

A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 bytes each. Find out

- Total main memory capacity
- Total cache memory capacity
- Total number of sets in the cache
- Number of bits for TAG, SET and word

Problem 2 (1/2)

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

Find out:

1) # bits to address main memory
2) # cache lines
3) # sets
4) Main Memory Address Format

Problem 2 (2/2)

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

Find out:

1) # bits to address main memory
2) # cache lines
3) # sets
4) Main Memory Address Format

Home work

A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses **word level addressing**. What is the address format? If we change the cache to a 4-way set associative cache, what is the new address format?

Replacement Algorithms (1/3)

Direct mapped cache

- No choice
- Each block maps to one line and replace that line

Replacement Algorithms (2/3)

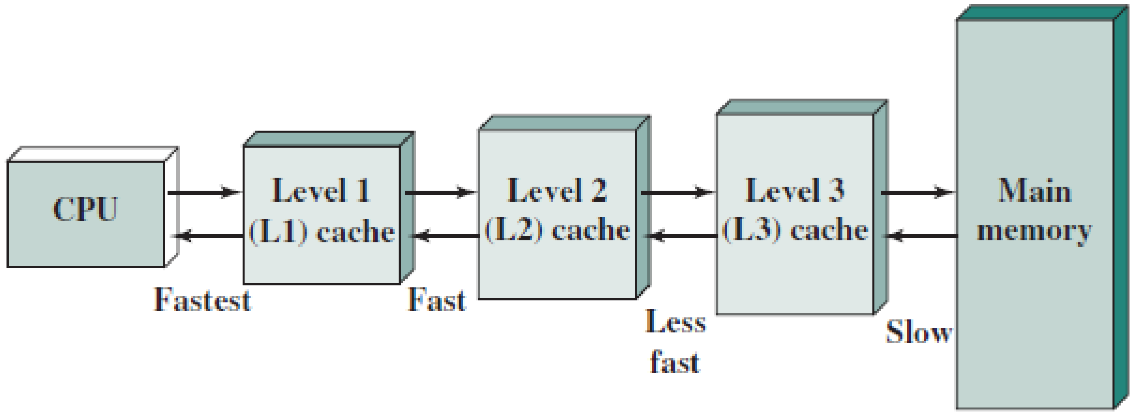
- Needed in Associative & Set Associative mapped cache
- Hardware implemented algorithm (speed)
- Methods:
  - Least Recently Used (LRU)
  - Least Frequently Used (LFU)
  - First In First Out (FIFO)
  - Random

Replacement Algorithms (3/3)

- **Least Recently used (LRU):** Replace the block that has been in the cache longest with no reference to it
  - e.g. 2 way set associative
  - Uses "USE" bits
  - Most effective method
- **Least frequently used:** Replace block which has had fewest hits
  - Uses counter with each line
- **First in first out (FIFO):** Replace block that has been in cache longest
  - Round robin or circular buffer technique
- **Random**

Issues with Writes

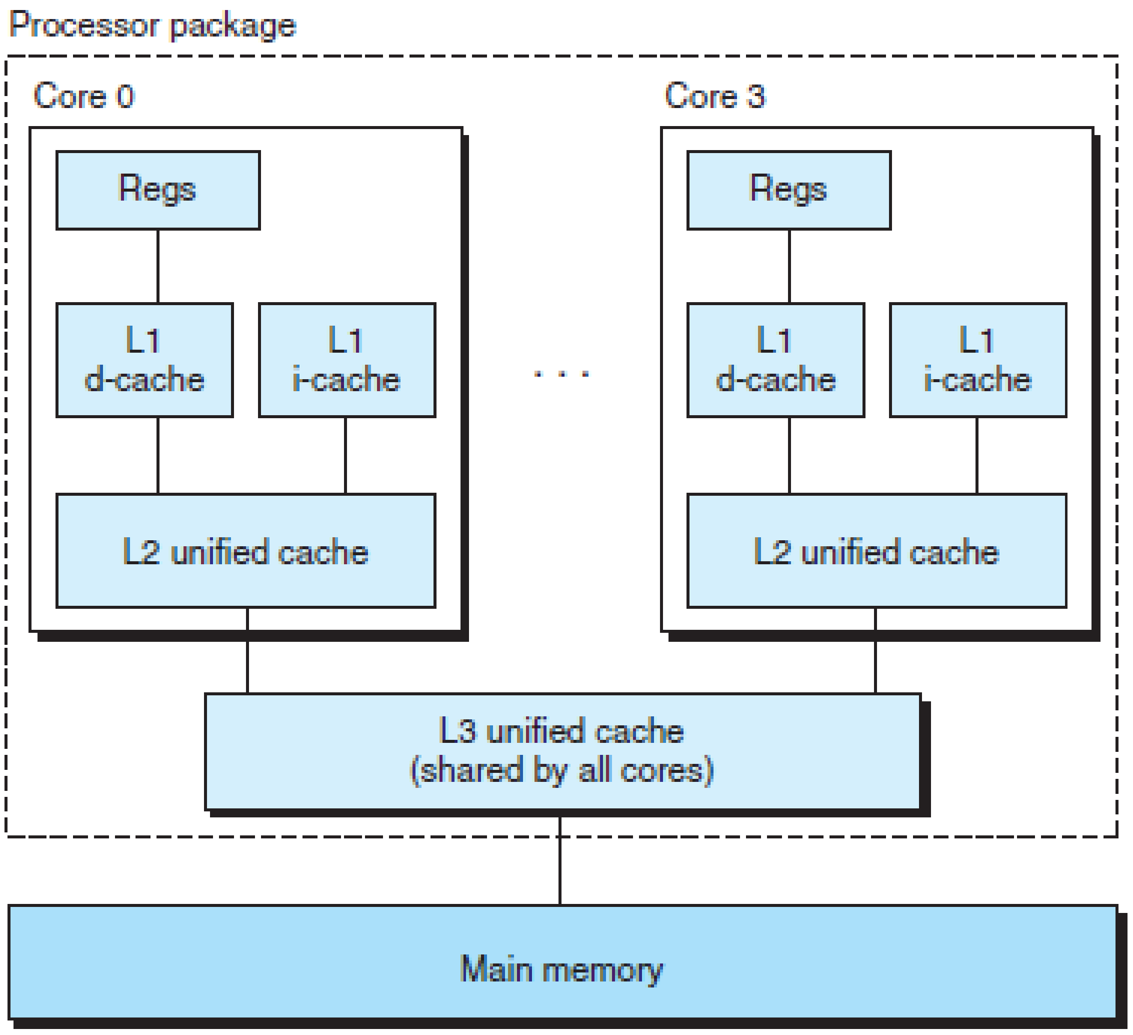
- Multiple copies of data exist:
  - L1, L2, L3, Main Memory, Disk
- What to do on a write-hit?
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (line different from



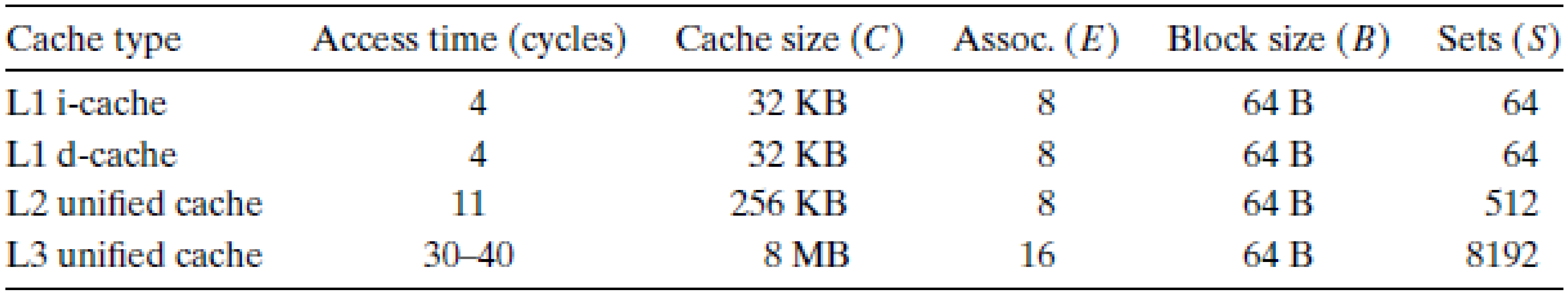
(b) Three-level cache organization

Cache and Main Memory

Intel Core i7 Cache Hierarchy



Intel Core i7 Cache Hierarchy



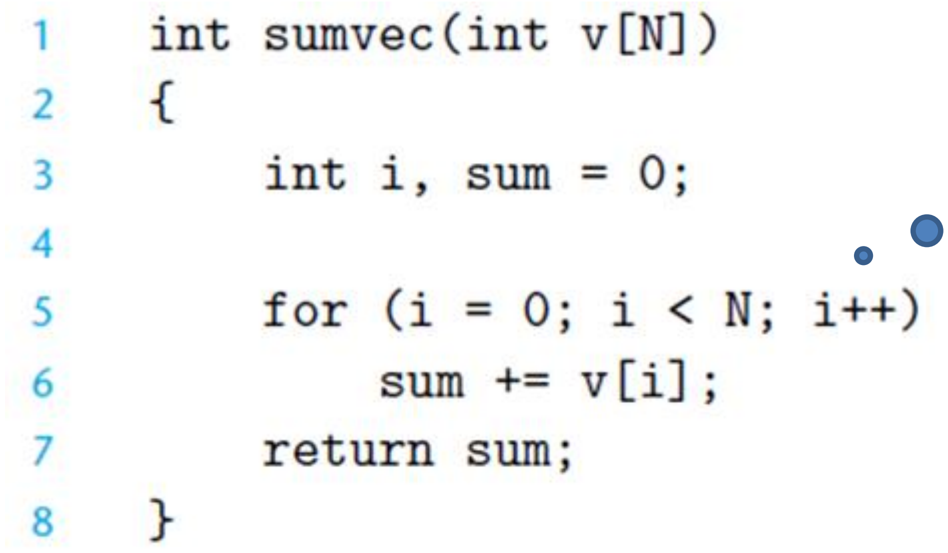
Cache type	Access time (cycles)	Cache size (C)	Assoc. (E)	Block size (B)	Sets (S)
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	11	256 KB	8	64 B	512
L3 unified cache	30–40	8 MB	16	64 B	8192

Characteristics of the Intel Core i7 cache hierarchy.

Performance Impact of Cache Parameters

- Associativity :
  - higher associativity ➔ more complex hardware
  - Higher Associativity ➔ Lower miss rate
  - Higher Associativity ➔ reduces average memory access time (AMAT)
- Cache Size
  - Larger the cache size ➔ Lower miss rate
  - Larger the cache size ➔ reduces average memory access time (AMAT)
- Block Size / Line Size:
  - Smaller blocks do not take maximum advantage of spatial locality.

Revisiting Locality of reference



```
int sumvec(int v[N])
{
    int i, sum = 0;

    for (i = 0; i < N; i++)
        sum += v[i];
    return sum;
}
```

Does this function have good locality?

N=8								
Address	0	1	2	3	4	5	6	7
Contents	V[0]	V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]
Access Order	1	2	3	4	5	6	7	8

Stride k – reference pattern

Byte Addressable memory and word length is 1 byte

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 1

$$\text{Stride} = \frac{\text{Address difference}}{\text{Word Length}}$$

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 2

Stride k – reference pattern

Byte Addressable memory and word length is 2 bytes

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

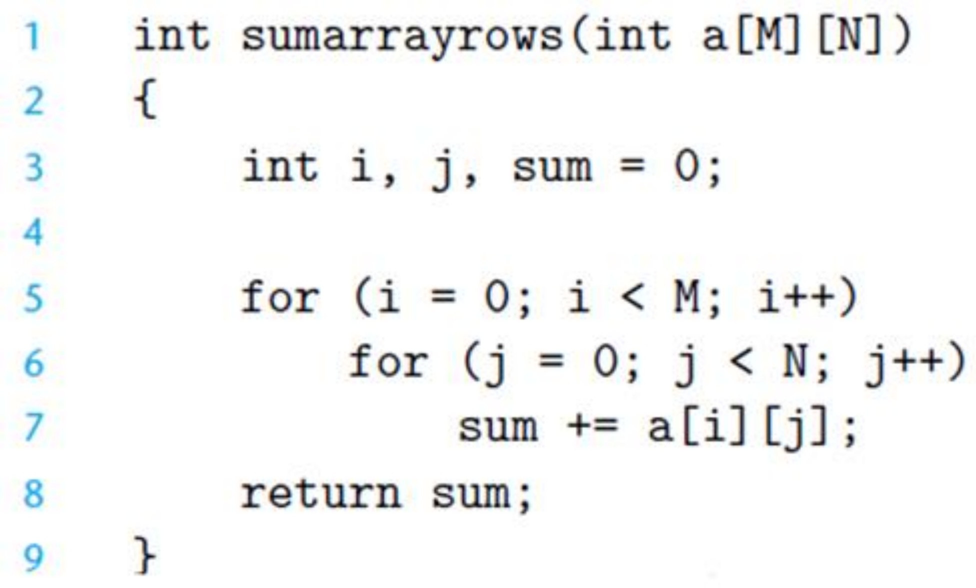
$\underbrace{0000, 0002}$ **Stride 1**

$$\text{Stride} = \frac{\text{Address difference}}{\text{Word Length}}$$

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

$\underbrace{0000, 0002, 0004}$ **Stride 2**

Revisiting Locality of reference



```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Does this function have good locality?

M = 2, N=3						
Address	0	1	2	3	4	5
Contents	A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]
Access Order	1	2	3	4	5	6

Revisiting Locality of reference

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Does this function have good locality?

M = 2, N=3						
Address	0	1	2	3	4	5
Contents	A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]
Access Order	1	3	5	2	4	6

Writing Cache Friendly Code

- Make the common case go fast
  - Focus on the inner loops of the core functions

- Minimize the misses in the inner loops
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

Example 1

```
int sumarrayrows(int a[4][4])
{
 int i, j, sum = 0;
 for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
     sum += a[i][j];
 return sum;
}
```

Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores array elements in row-major order

Example 1(Contd..)

```
int sumarrayrows(int a[4][4])
{
 int i, j, sum = 0;
 for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
     sum += a[i][j];
 return sum;
}
```

L0

L1

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0				
i = 1				
i = 2				
I = 3				

W0

W1

W2

W3

W4

W5

W6

W7

W8

W9

W10

W11

W12

W13

W14

W15

Example 2:

```
int sumarraycols(int a[4][4])
{
 int i, j, sum = 0;
 for (j = 0; j < 4; j++)
   for (i = 0; i < 4; i++)
   sum += a[i][j];
 return sum;
}
```

Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

Example 1(Contd..)

```
int sum_array(int a[4][4])
{
 int i, j, sum = 0;
 for (j = 0; j < 4; j++)
   for (i = 0; i < 4; i++)
   sum += a[i][j];
 return sum;
}
```

L0
L1

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0				
i = 1				
i = 2				
I = 3				

W0
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15

Home Work – Which one is better ?

Program 1:

```
for (int i = 0; i < n; i++) {
    z[i] = x[i] - y[i];
    z[i] = z[i] * z[i];
}
```

Program 2:

```
for (int i = 0; i < n; i++) {
        z[i] = x[i] - y[i];
}
for (int i = 0; i < n; i++) {
        z[i] = z[i] * z[i];
}
```