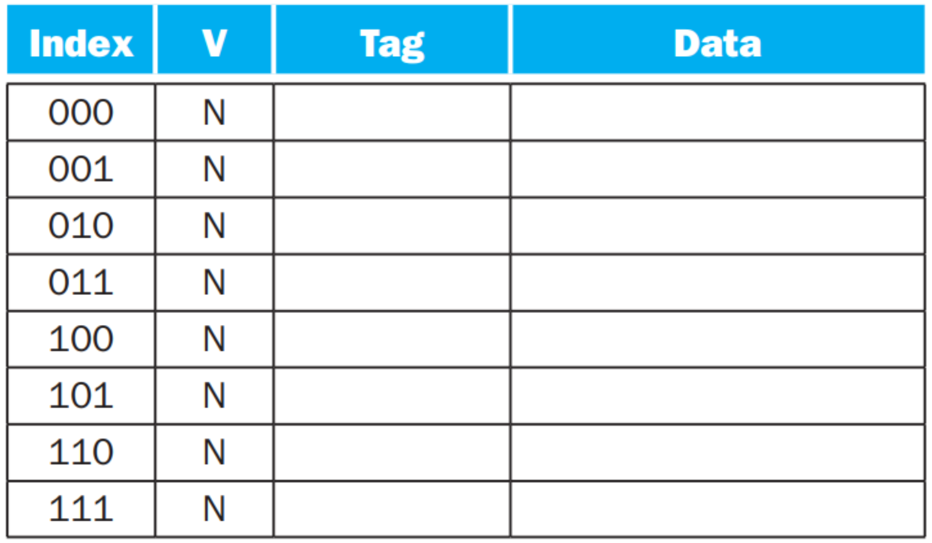
# Computer Architecture
(ENE1004)

Lec – 21: Large and Fast: Exploiting Memory Hierarchy (Chapter 5) – 3

## Schedule

- Final exam: Jun. 19, Monday
  - (24334)1:25~2:25pm
  - (23978) 2:30~3:30pm
  - Sample questions will be provided by Jun. 17 (Saturday)
- Assignment #2: Jun. 20, Tue. by midnight
  - It is put back as much as possible
- Remaining class days
  - 5(Mon), 8(Thur), 12(Mon), 15 (Thur)

# Access a Direct Mapped Cache (1)



| Index | V | Tag | Data |
|---|---|---|---|
| 000 | N | | |
| 001 | N | | |
| 010 | N | | |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | N | | |
| 111 | N | | |

a. The initial state of the cache after power-on

- Each entry of the cache consists of “index” + “valid bit” + “tag” + “data”
    - There are 8 entries (blocks) in the cache, each of which is identified based on the 3-bit “index”
    - The 1-bit “valid bit” tells whether the information in the entry is valid or not
    - If a data is placed in its corresponding entry, its 2-bit “tag” is set to indicate the specific data
- At first, the cache is empty (it does not hold any data) right after power-on
    - The valid bit of each entry is set to “N”
    - The tag and data are also empty

# Access a Direct Mapped Cache (2)

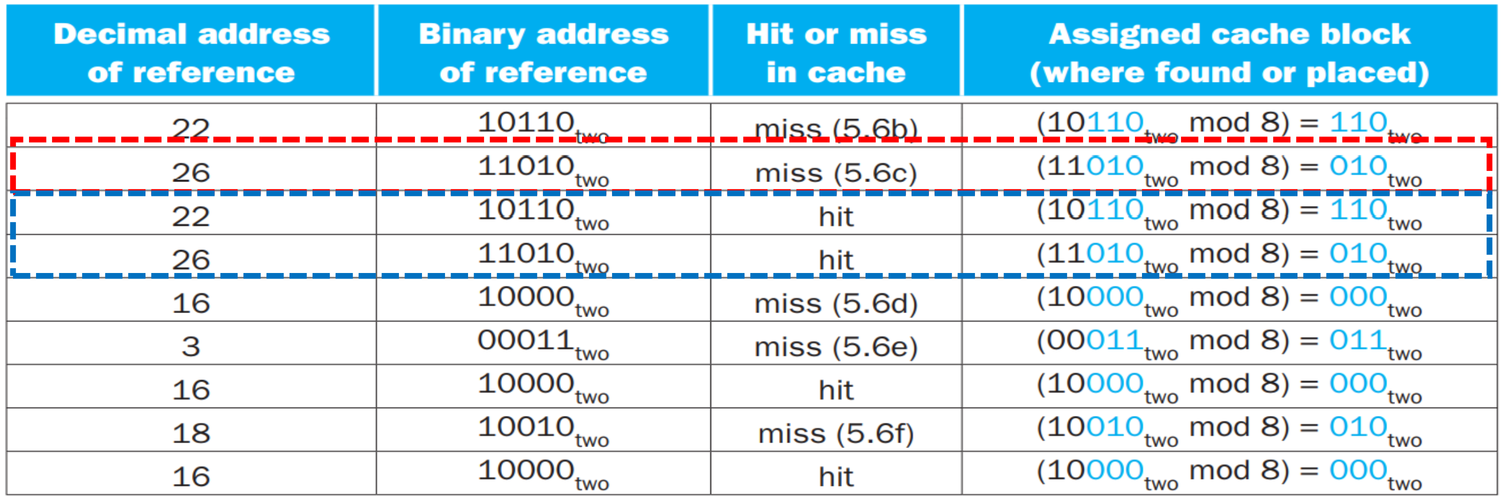
| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | N | | |
| 001 | N | | |
| 010 | N | | |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

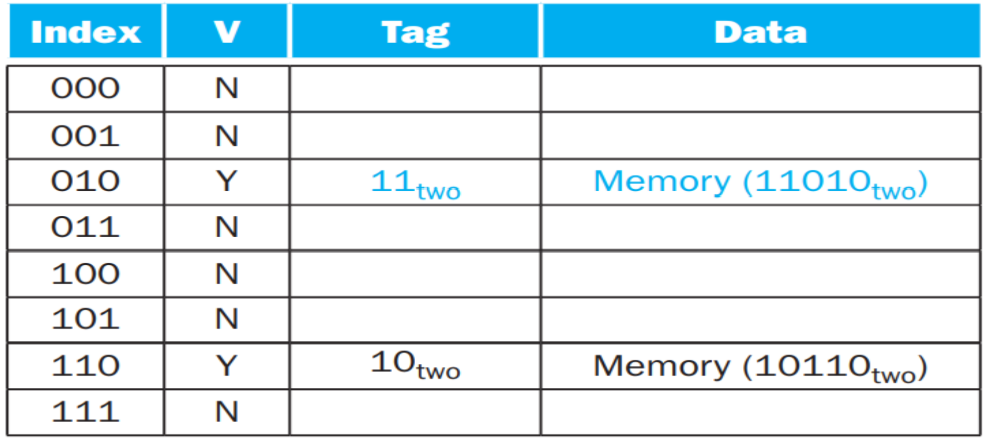
b. After handling a miss of address ($10110_{two}$)

- An example scenario where a series of requests (references) are given to the cache
  - If the referenced data is in the cache, it is “hit” and the data is serviced from the cache
  - If the referenced data is not in the cache, it is “miss” and the existing data is replaced with the referenced data, which is copied from the main memory
- Data 22 ($10110_{two}$) is requested
  - It can be placed in the cache block whose “index” is $110_{two}$ (the lower 3 bits $10110_{two}$)
  - It is “miss” because the valid bit is set to N (Data 22 does not exist in the cache block)
  - Data 22 is brought from the main memory and stored in the cache block whose index is $110_{two}$
  - Tag is set to $10_{two}$ to indicate data 22 (the upper 2 bits of $10110_{two}$); valid bit is set to Y

# Access a Direct Mapped Cache (3)



| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |

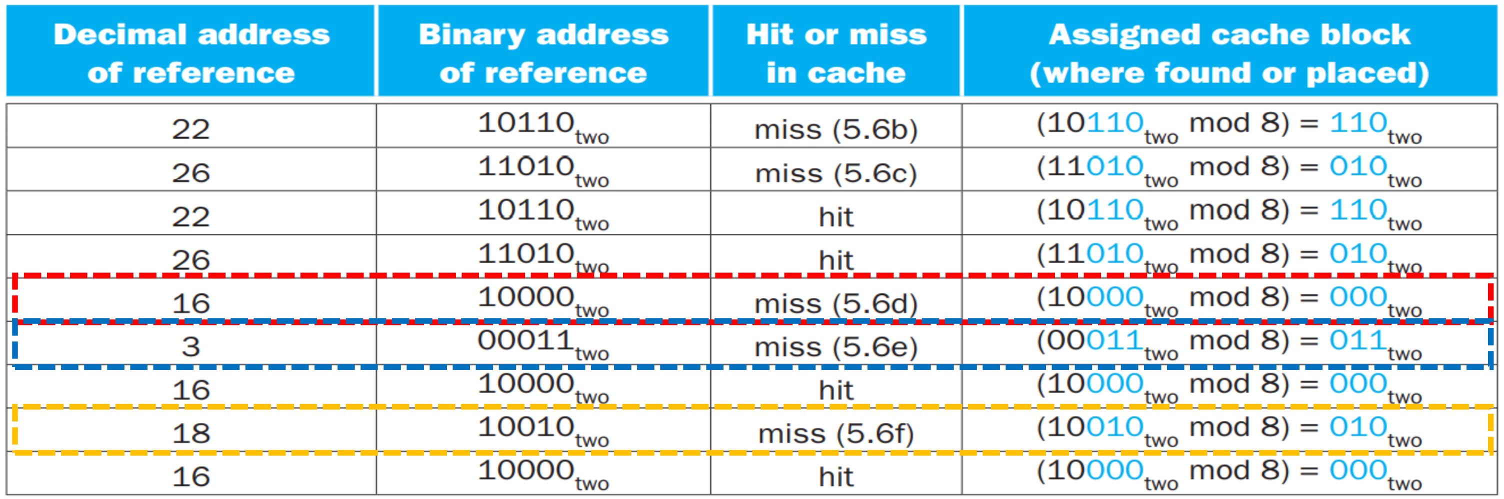


| Index | V | Tag | Data |
|---|---|---|---|
| 000 | N | | |
| 001 | N | | |
| 010 | Y | $11_{two}$ | Memory ($11010_{two}$) |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

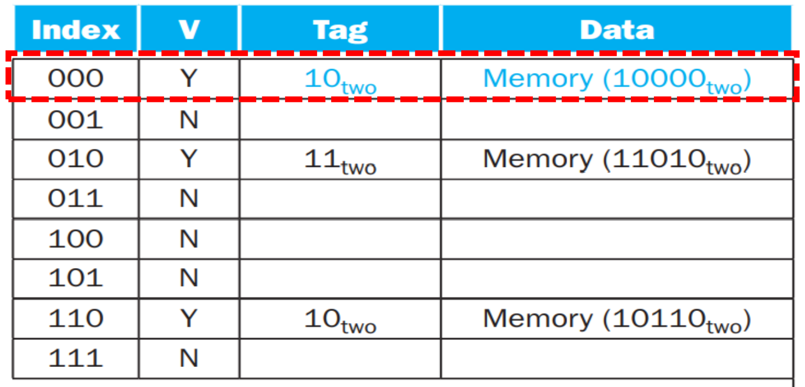
c. After handling a miss of address ($11010_{two}$)

- Data 26 ($11010_{two}$) is requested
  - It can be placed in the cache block whose "index" is $010_{two}$ (the lower 3 bits $11010_{two}$)
  - It is "miss" because the valid bit is set to N (Data 26 does not exist in the cache block)
  - Data 26 is brought from the main memory and stored in the cache block whose index is $010_{two}$
  - Tag is set to $11_{two}$ to indicate data 26 (the upper 2 bits of $11010_{two}$); valid bit is set to Y
- Data 22 ($10110_{two}$) is requested again
  - Go to the index $110_{two}$; first, check its valid bit Y; then, check the tag $10_{two}$ ; it is "hit"
- Data 26 ($11010_{two}$) is requested again
  - Go to the index $010_{two}$; first, check its valid bit Y; then, check the tag $11_{two}$ ; it is "hit"

# Access a Direct Mapped Cache (4)

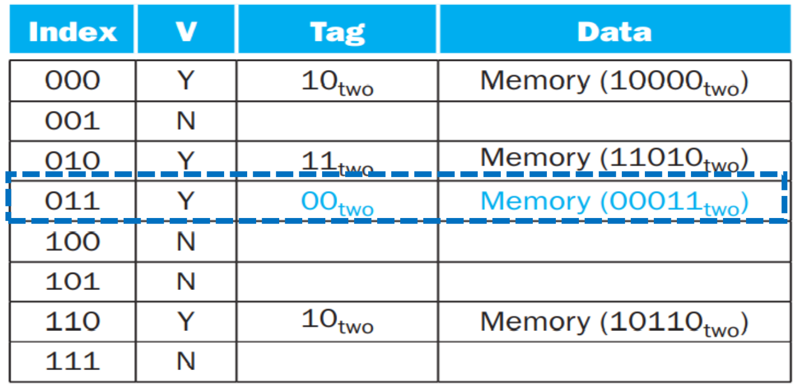


| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |



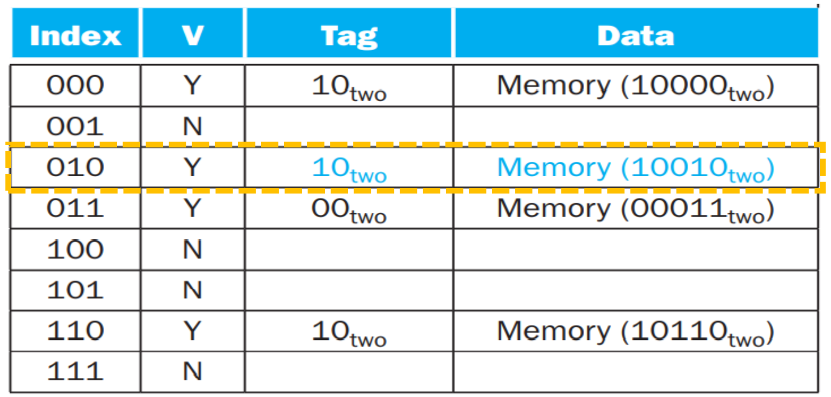
| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | N | | |
| 010 | Y | $11_{two}$ | Memory ($11010_{two}$) |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

d. After handling a miss of address ($10000_{two}$)



| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | N | | |
| 010 | Y | $11_{two}$ | Memory ($11010_{two}$) |
| 011 | Y | $00_{two}$ | Memory ($00011_{two}$) |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

e. After handling a miss of address ($00011_{two}$)

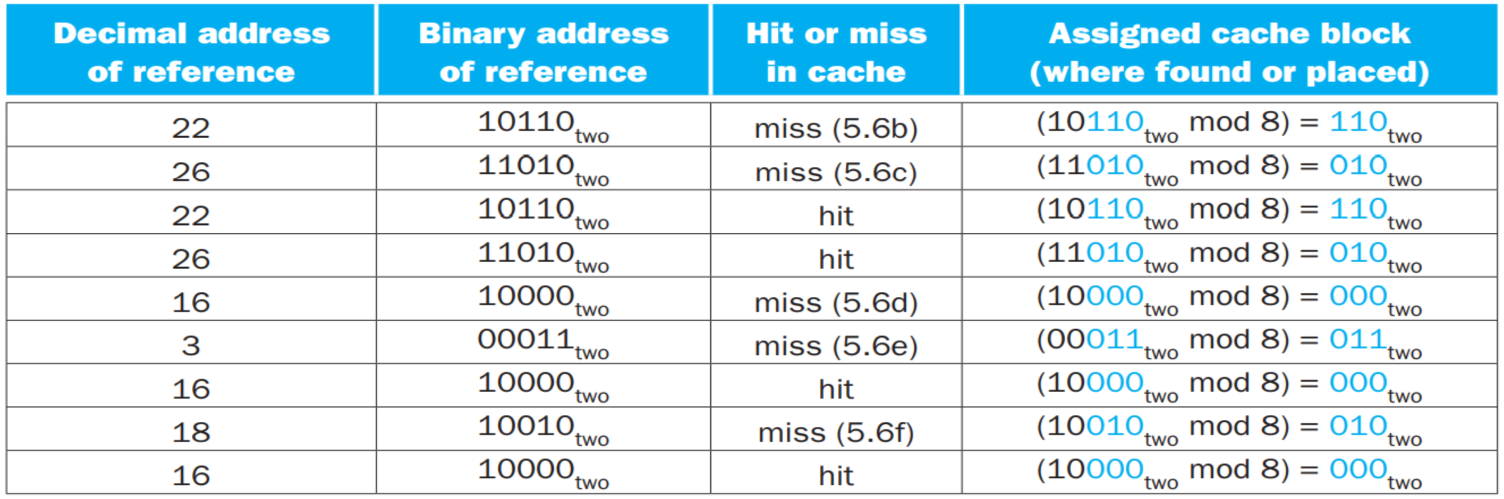


| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | N | | |
| 010 | Y | $10_{two}$ | Memory ($10010_{two}$) |
| 011 | Y | $00_{two}$ | Memory ($00011_{two}$) |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

f. After handling a miss of address ($10010_{two}$)

- 26($11010_{two}$) in cache block (index 010) is replaced by 18($10010_{two}$)
  - They are mapped to the same cache block (index 010)
  - The recently referenced data item replaces less recently referenced data
  - This behavior allows a cache to take advantage of "temporal locality"

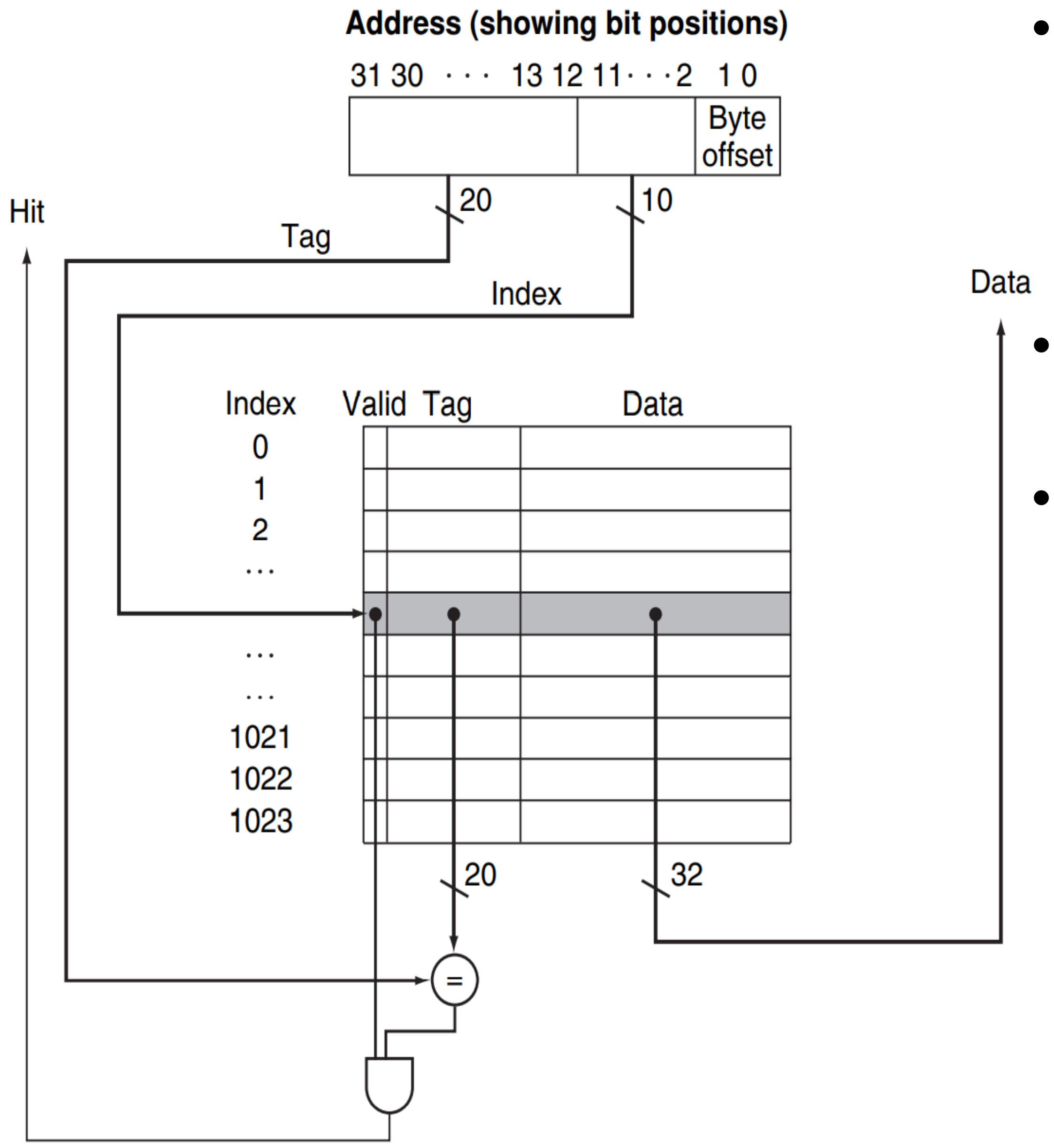
# Access a Direct Mapped Cache (5)



| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |

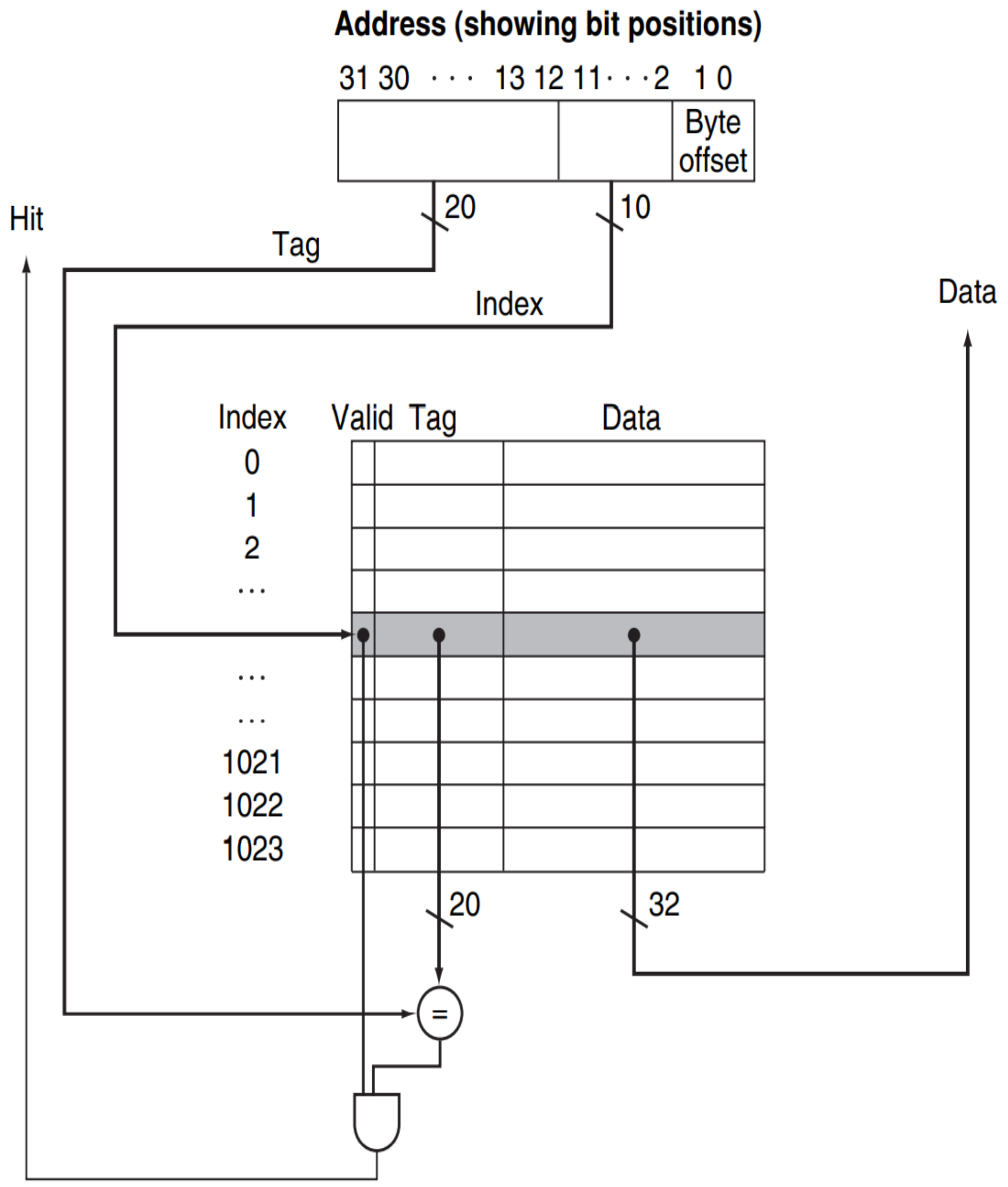
- In this example, how many cache accesses are there?
  - 9 cache accesses
- Among them, how many hits and misses are there?
  - 4 hits and 5 misses
- What is the cache hit ratio (miss ratio)?
  - 4/9 (5/9)
- An ideal cache achieves the hit ratio of “1”

# Direct Mapped Cache for the Real World (1)



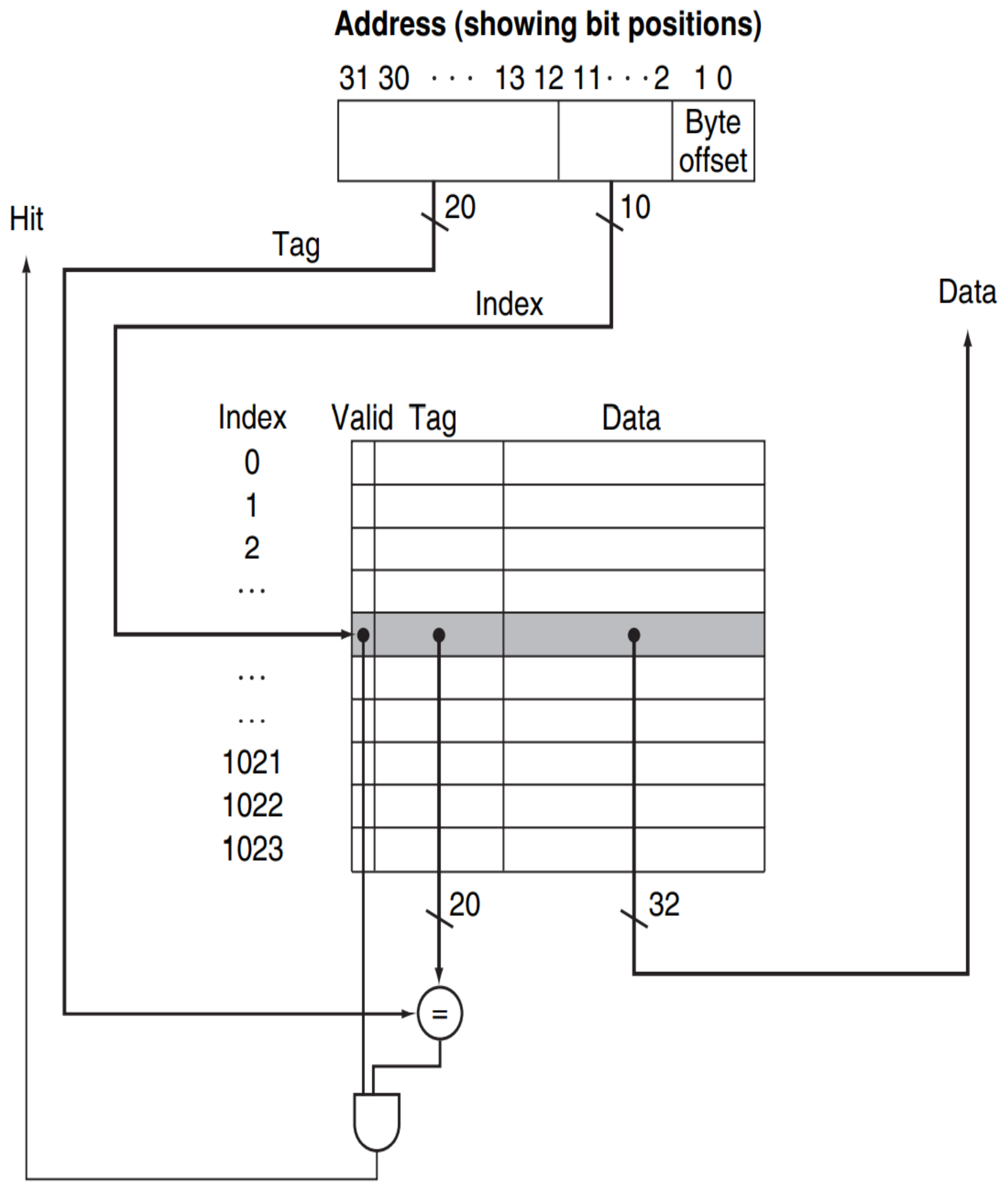
- Assumption
  - 32-bit address (vs 5-bit address)
  - The cache size is $2^{10}$ blocks (vs 8 blocks)
  - The block size is $2^2$ bytes
- Now, let us determine how an address of data is used for a cache access
- Byte offset
  - Each cache block can contain $2^2$ bytes of data
  - The base unit of a data item is a "byte"
  - We need to identify a byte of data within cache block
  - The last 2 bits are used for byte offset
  - A byte of data is referenced, $2^2$ bytes of data including the referenced and neighboring data are brought from the memory to the cache
  - This behavior allows a cache to take advantage of "spatial locality"

# Direct Mapped Cache for the Real World (2)



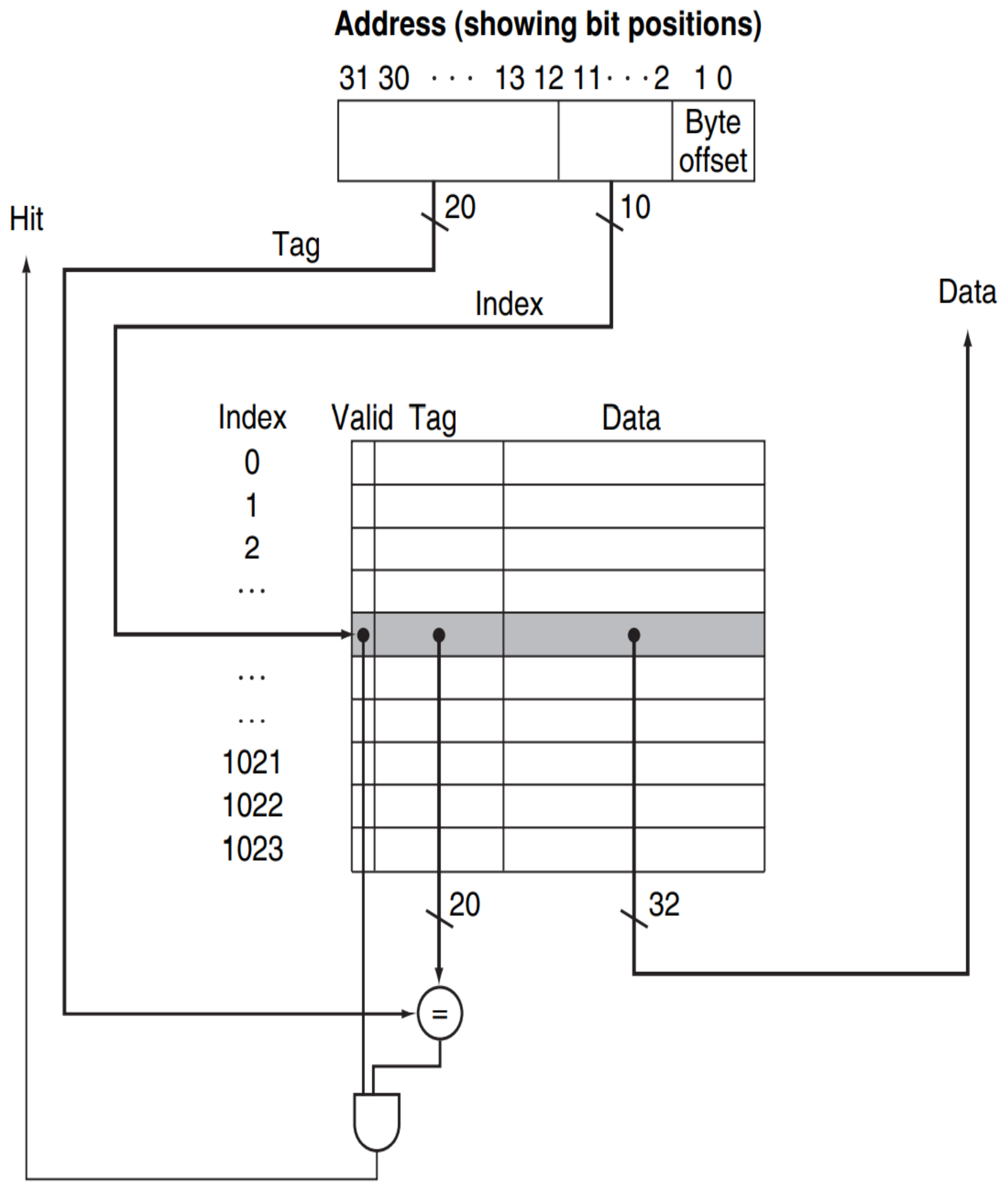
- Assumption
  - 32-bit address (vs 5-bit address)
  - The cache size is $2^{10}$ blocks (vs 8 blocks)
  - The block size is $2^2$ bytes
- Index (to select a cache block)
  - There are $2^{10}$ blocks in the cache
  - We need "10 bits" to determine the index of the data
- Tag (to identify a specific data item)
  - The remaining "20 bits" are used for the tag
  - 20 = 32 – 10 (for index) – 2 (for byte offset)
- Cache access
  - Given an address, the target cache block is accessed using the 10-bit (b11-b2) index value
  - Then, the 20-bit (b31-b12) tag value is compared to the tag field of the cache block
  - If they are equal ("hit"), the entire 4-byte data is read
  - If they are not equal ("miss"), the referenced 4-byte data replaces the existing 4-byte data in the cache

# Generalization of Direct Mapped Cache (1)



- Assumption
  - 32-bit address
  - The cache size is $2^n$ blocks
  - The block size is $2^m$ words ($2^{m+2}$ bytes)
- Byte offset
  - Each cache block can contain $2^{m+2}$ bytes of data
  - "m bits" are used to identify a word within a block
  - "2 bits" are used to identify a byte within a word
  - The last m+2 bits are used for byte offset
- Index (to select a cache block)
  - There are $2^n$ blocks in the cache
  - We need "n bits" to determine the index of the data
- Tag (to identify a specific data item)
  - The remaining bits are used for the tag
  - The size of the tag filed is $32 - (n + m + 2)$

# Generalization of Direct Mapped Cache (2)

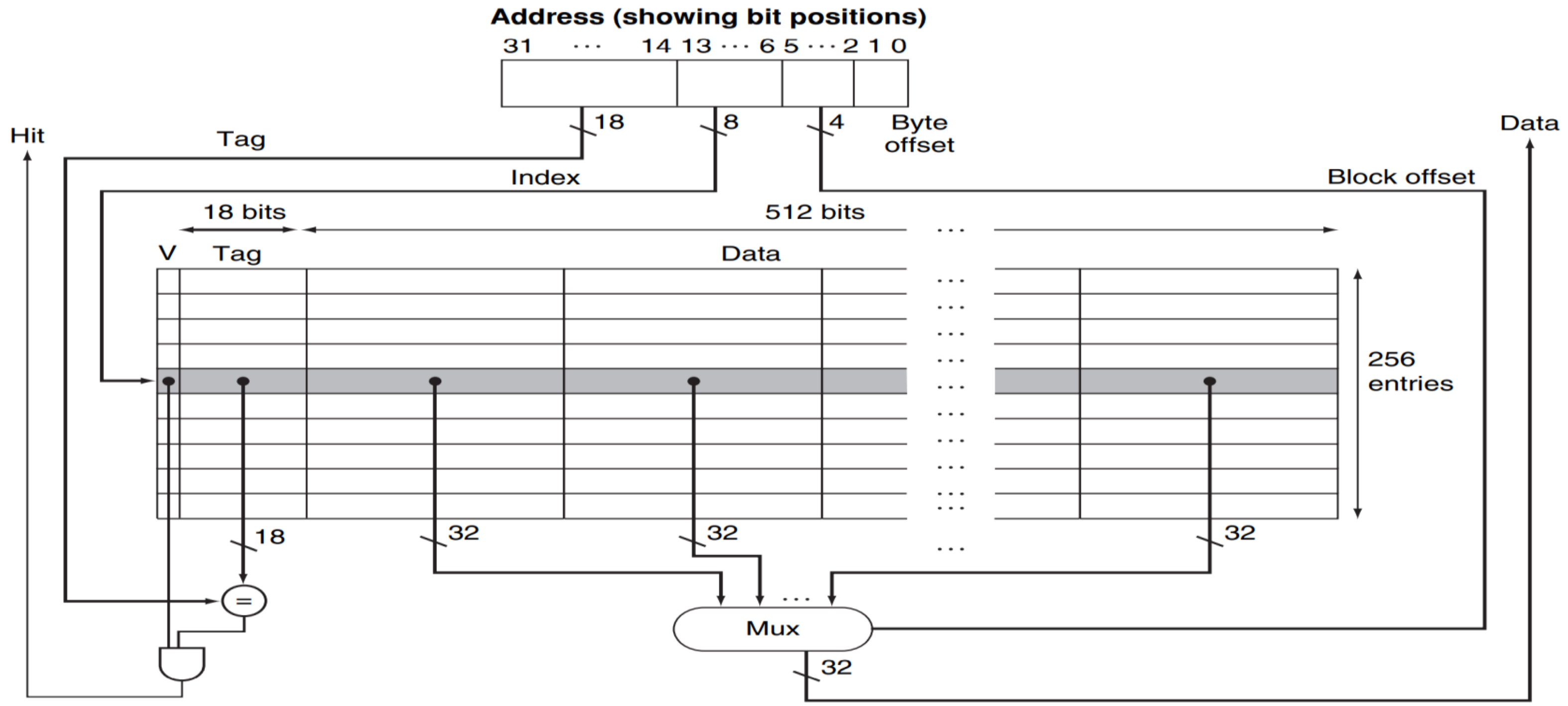


- Assumption
  - 32-bit address
  - The cache size is $2^n$ blocks
  - The block size is $2^m$ words ($2^{m+2}$ bytes)
- Address fields
  - Byte offset: m+2 bits
  - Index: n bits
  - Tag: 32 – (n + m + 2) bits
- The total number of bits in a cache
  - # of blocks x (block size + tag size + valid bit size)
  - $2^n$ x ($2^{m+2}$ bytes + (32-n-m-2) bits + 1 bit)
  - $2^n$ x ($2^{m+5}$ + (32-n-m-2) + 1) bits
- Due to the tag and valid bit fields, the actual size of the cache is larger than the total amount of data ($2^n$ x $2^{m+2}$ bytes)

# Bits in Cache

- Question: How many total bits are required for a direct mapped cache with <u>16 KB of data</u> and <u>4-word blocks</u>, assuming <u>a 32-bit address</u>?
- Hint: We need to calculate "total # of blocks (indices)" and "# bits for tag"
    - Cache size = total # blocks x (block size + tag size + valid bit size) for each block
- The total number of cache blocks
    - The entire size of blocks = 16 KB ($2^{14}$ Bytes = $2^{12}$ words)
    - The size of each block = 4 words ($2^{4}$ Bytes = $2^{2}$ words)
    - The total number of blocks = entire size / a block size = $2^{10}$ = 1024
    - The number of bits for "index" is 10 bits
- The number of bits for "tag"
    - # bits for tag = total # address bits – (# bits for index + # bits for byte offset)
    - # bits for byte offset = 4, because the size of a cache block is of 4-word ($2^{4}$ Bytes)
    - # bits for tag = 32 – (10 + 4) = 18
- Cache size = total # blocks x (block size + tag bit size + valid bit size) for each block
    - Cache size = 1024 x (4 words + 18 bits + 1 bit) = 1024 x ($2^{7}$ bits + 18 bits + 1 bit) = $2^{10}$ x 147 bits
    - 147 Kbits

# Bits in Cache (2)



- Cache block size: 512 bits = 64 ($2^6$) Bytes = 16 words
  - Byte offset = 6 bits (b5-b0)
  - You may want to access the data in the unit of “word”; then, you can use upper bits in the byte offset bits; in this example, upper 4 bits (b5-b2) is used for word offset (called “block offset”)
- Total number of cache blocks is 256 ($2^8$); # bits for index is 8 (b13-b6)
- # bits for tag = 32 – (8 + 6) = 18 bits
- The cache size = 256 x (512 bits for data + 18 bits for tag + 1 bit for valid bit)