EC-252: COMPUTER ARCHITECTURE AND MICROPROCESSORS

Vaskar Raychoudhury

Indian Institute of Technology Roorkee

Cache Memory

- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced,
- Thus reducing the total execution time of the program
- Such a fast small memory is referred to as cache memory
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component

Cache Memory

- When CPU needs to access memory, the cache is examined
- If the word is found in the cache, it is read from the fast memory
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word

Cache Memory

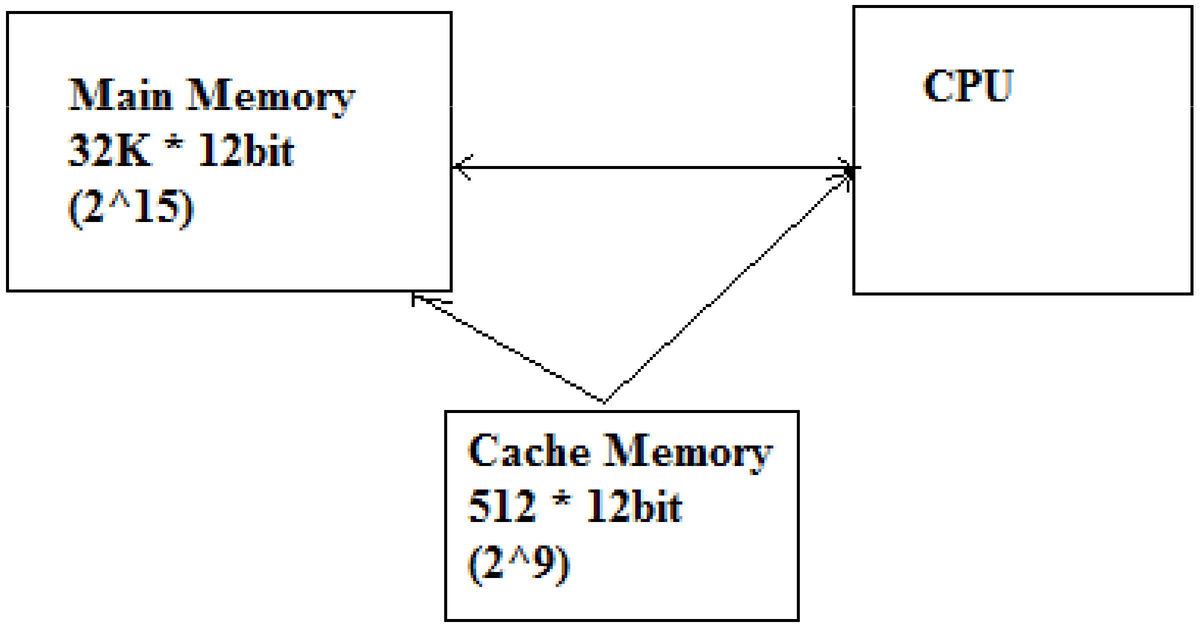
- The performance of cache memory is frequently measured in terms of a quantity called **hit ratio**
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**
- Otherwise, it is a **miss**
- **Hit ratio = hit / (hit+miss)**

Cache Memory

- The basic characteristic of cache memory is its fast access time,
- Therefore, very little or no time must be wasted when searching the words in the cache
- The transformation of data from main memory to cache memory is referred to as a **mapping** process, there are three types of mapping:
  - Associative mapping
  - Direct mapping
  - Set-associative mapping

Cache Memory

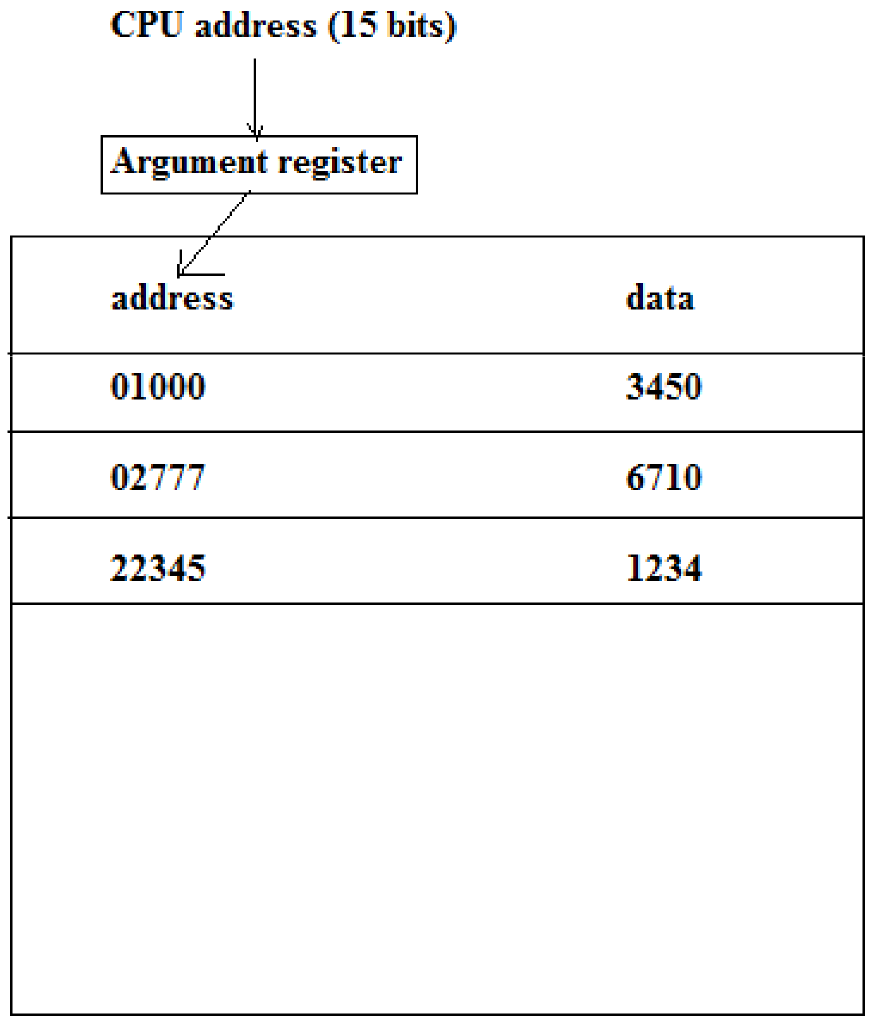
- To help understand the mapping procedure, we have the following example:



Associative Mapping

- The fastest and most flexible cache organization uses an associative memory
- The associative memory stores both the address and data of the memory word
- This permits any location in cache to store any word from main memory
- The address value of 15 bits is shown as a five-digit **octal** number and its corresponding 12-bit word is shown as a four-digit octal number

Associative Mapping



address	data
01000	3450
02777	6710
22345	1234

Associative Mapping

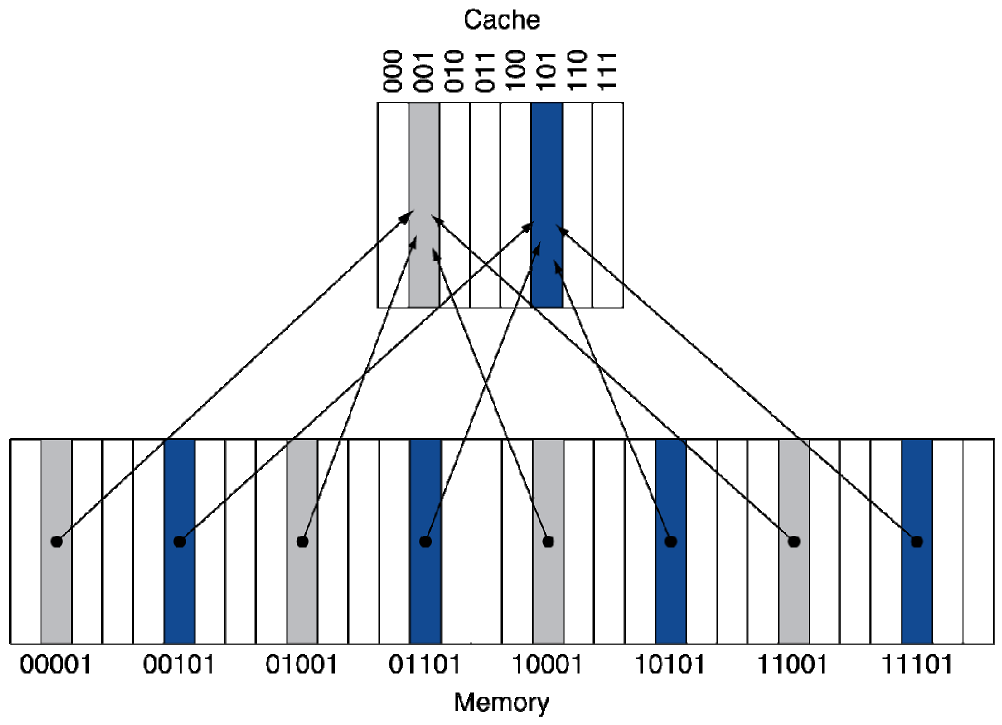
- A CPU address of 15 bits is places in the argument register and the associative memory is searched for a matching address
  - If the address is found, the corresponding 12-bits data is read and sent to the CPU
  - If not, the main memory is accessed for the word
- If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache

Direct Mapping

- Associative memory is expensive compared to RAM
- In general case, there are 2^k words in cache memory and 2^n words in main memory (in our case, k=9, n=15)
- The n bit memory address is divided into two fields: k-bits for the index and n-k bits for the tag field

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)

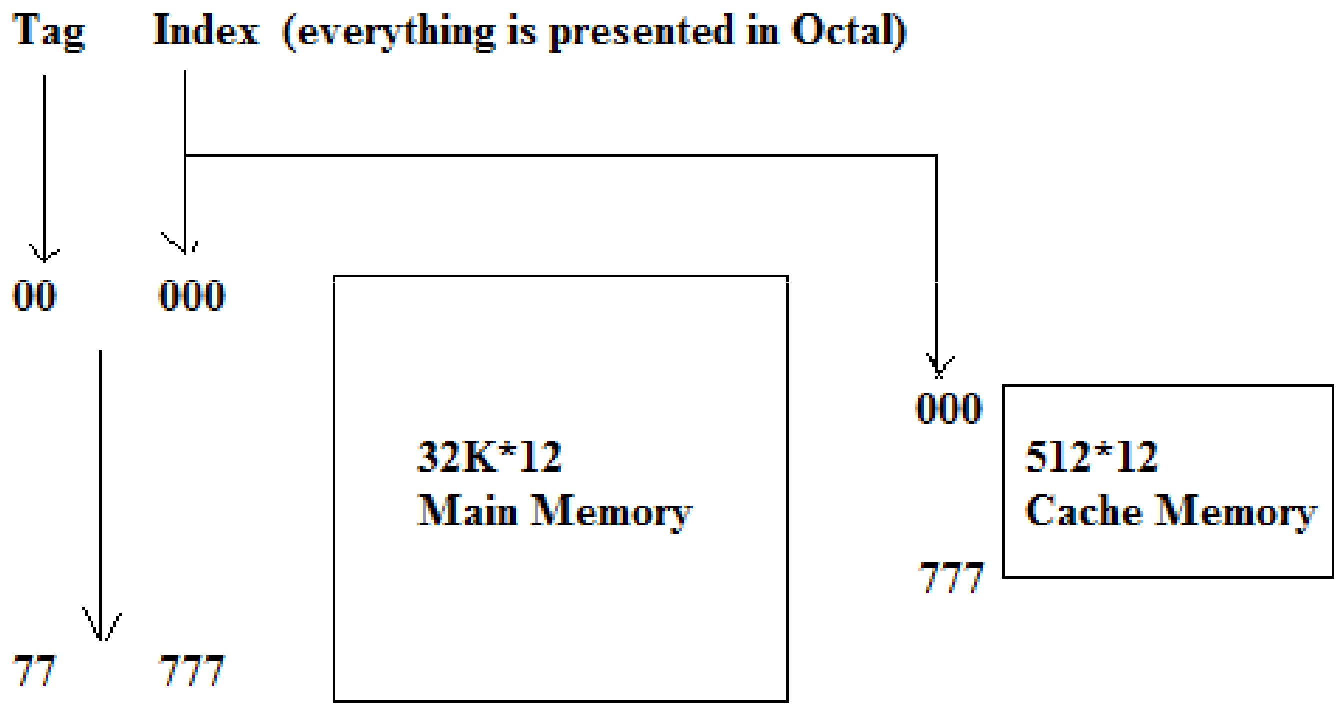


- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Direct Mapping



Direct Mapping

Memory Address	Memory Data
00000	1220
00777	2340
01000	3450
01111	2222
01777	4560
02000	5670
02777	6710

Index Address	Tag	Data
000	00	1220
111	01	2222
777	02	6710

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

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

Cache Example (2)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (3)

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (4)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (5)

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

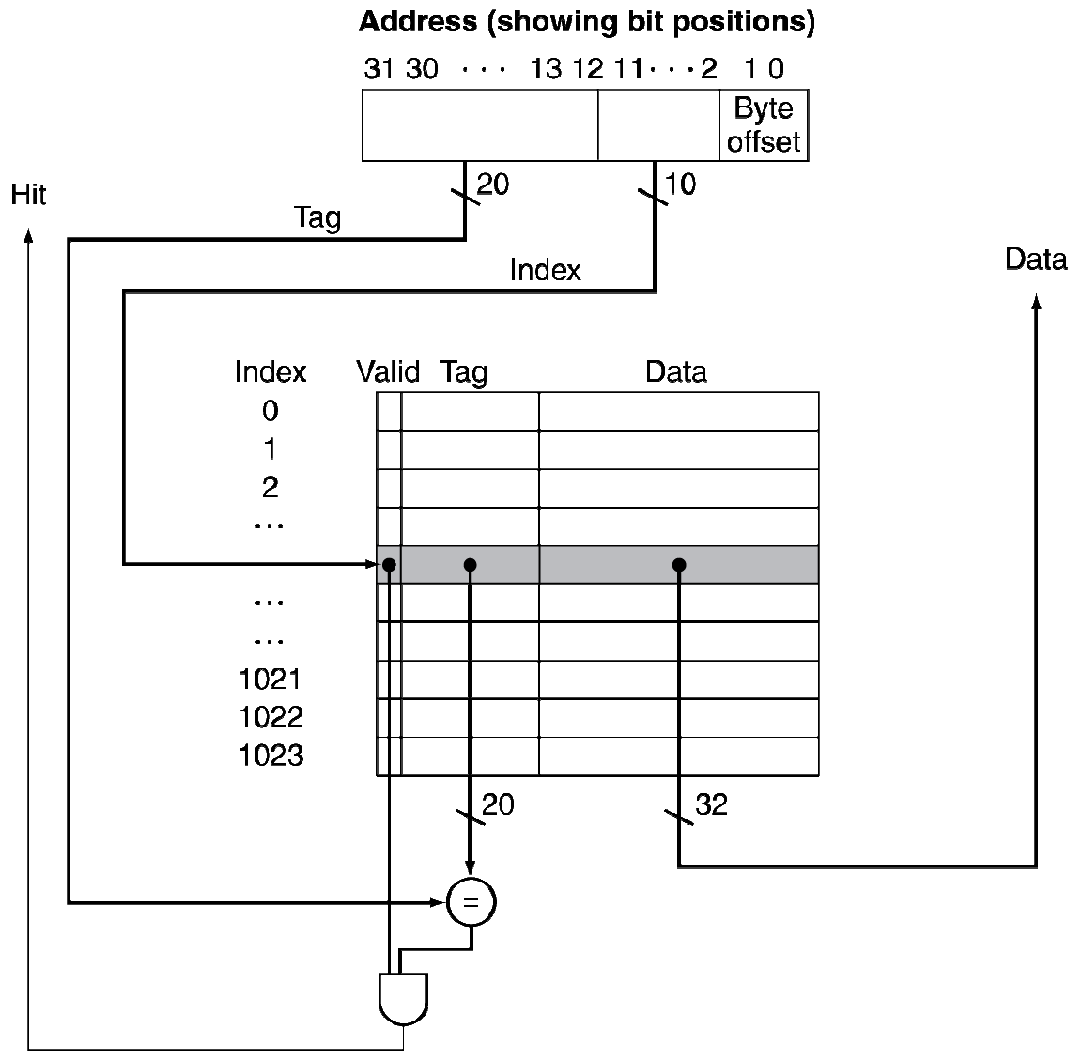
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (6)

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision



Set-Associative Mapping

- The disadvantage of direct mapping is that
  - two words with the same index in their address but with different tag values cannot reside in cache memory at the same time
- Set-Associative Mapping is an improvement over the direct-mapping in that
  - each word of cache can store two or more word of memory under the same index address

Set-Associative Mapping

Memory Address	Memory Data
00000	1220
00777	2340
01000	3450
01111	2222
01777	4560
02000	5670
02777	6710

Index Address	Tag	Data	Tag	Data
000	01	3450	02	5670
111	01	2222		
777	02	6710	00	2340

Set-Associative Mapping

- In the slide, each index address refers to two data words and their associated tags
- Each tag requires six bits and each data word has 12 bits, so the word length is 2*(6+12) = 36 bits