ACCESSING A CACHE

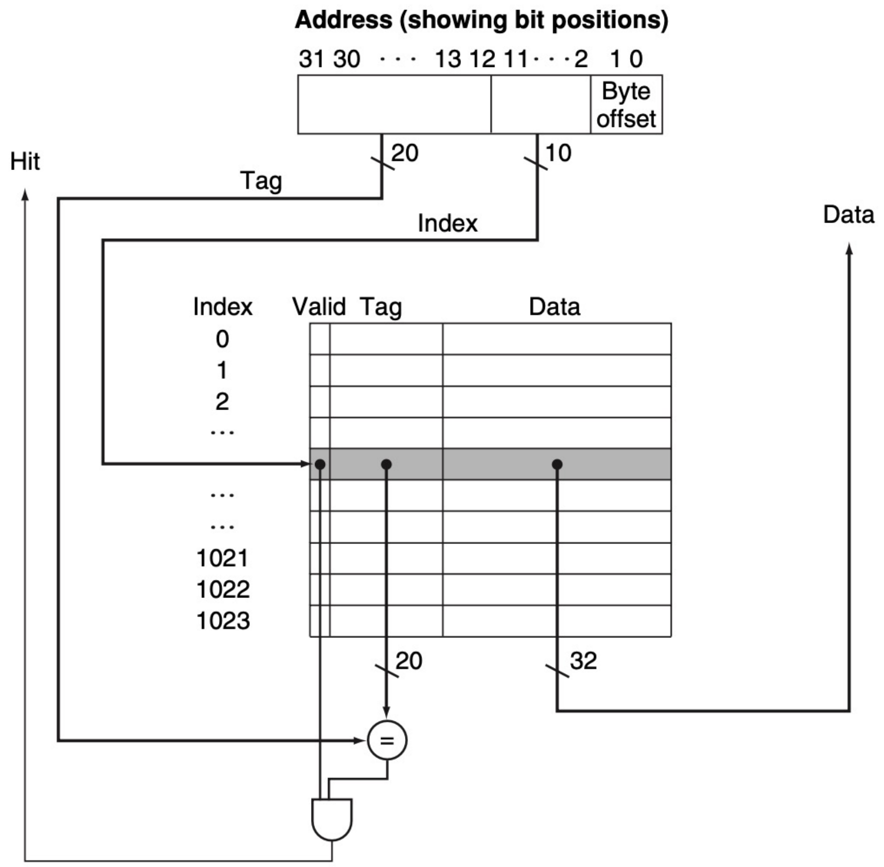
- To look in the cache: the low-order bits of an address is used to find the unique cache entry to which the address could map.

- ***tag field***, which is used to compare with the value of the tag field of the cache

- ***cache index***, which is used to select the block

Index + Tag = Memory address of word contained in the cache block.

- *n*-bit index field is used as an address to reference the cache. 2^n different values are possible, so the total number of entries in a direct-mapped cache must be a power of 2.

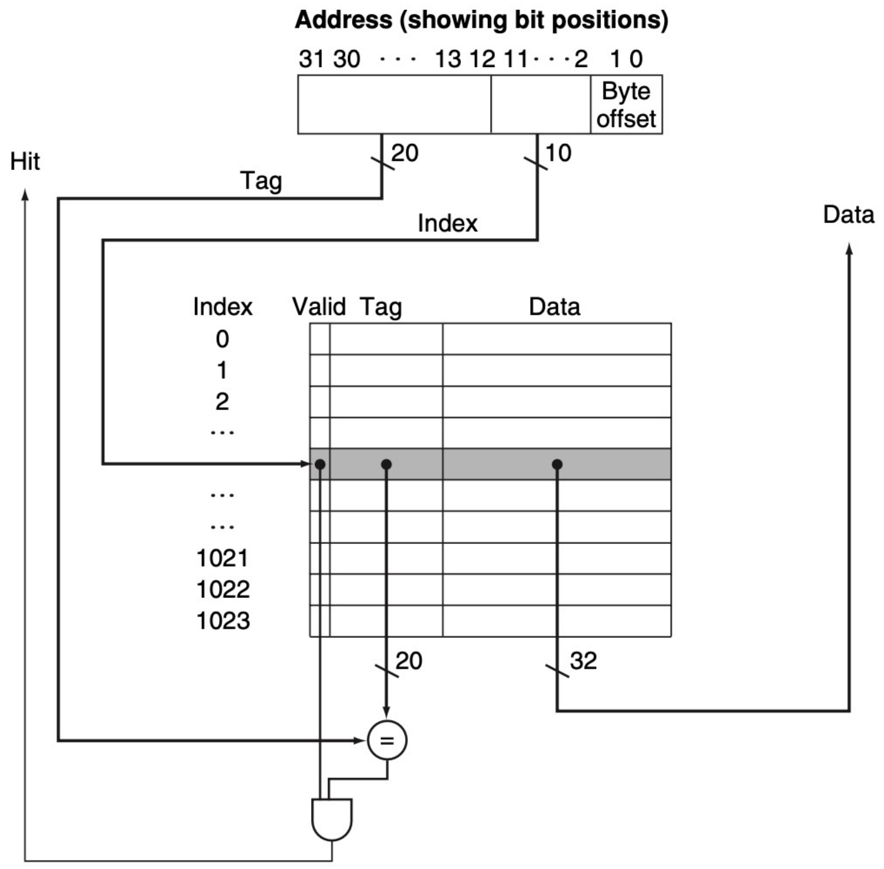


ACCESSING A CACHE

- In the MIPS architecture, words are aligned to multiples of four bytes, the least significant two bits of every address specify a byte within a word.

- Hence, the least significant two bits are ignored when selecting a word in the block.

The total number of bits needed for a cache is a function of the cache size and the address size, because the cache includes both the storage for the data and the tags.



ACCESSING A CACHE

For the following situation:

- 32-bit addresses
- A direct-mapped cache
- The cache size is 2^n blocks, so n bits are used for the index.
- The block size is 2^m words (2^{m+2} bytes), so m bits are used for the word within the block, and two bits are used for the byte part of the address.

The size of the tag field is $32 - (n + m + 2)$.

The total number of bits in a direct-mapped cache is

$$2^n \times (\text{block size} + \text{tag size} + \text{valid field size}).$$

ACCESSING A CACHE

Since the block size is 2^m words (2^{m+5} bits), and we need 1 bit for the valid field;

the number of bits in such a cache is

$$\mathbf{2^n \times (block\ size + tag\ size + valid\ field\ size)}$$

$$2^n \times (2^m \times 32 + (32 - (n + m + 2) + 1)$$

$$= 2^n \times (2^m \times 32 + 31 - n - m)$$

Although this is the actual size in bits, the naming convention is to exclude the size of the tag and valid field and to count only the size of the data.

ACCESSING A CACHE

Bits in a Cache

How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

We know that 16 KiB is 4096 (2^{12}) words.

With a block size of 4 words (2^2), there are $\frac{4096}{4}$ blocks = 1024 (2^{10}) blocks.

Each block has 4×32 or 128 bits of data plus a tag, which is $32 - 10 - 2 - 2$ bits, plus a valid bit.
Thus, the total cache size is

$$\mathbf{2^n \times (block\ size + tag\ size + valid\ field\ size)}$$

$$2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1)$$
$= 2^{10} \times 147 = 147$ Kibibits or 18.4 KiB for a 16 KiB cache.

ACCESSING A CACHE

Mapping an Address to a Multiword Cache Block

Consider a cache with 64 blocks and a block size of 16 bytes. To what block number does byte address 1200 map?

The block is given by **(Block address) modulo (Number of blocks in the cache)**

where the address of the block is $\frac{\text{Byte address}}{\text{Bytes per block}}$

Thus, with 16 bytes per block, byte address 1200 is block address

$$\left\lfloor \frac{1200}{16} \right\rfloor = 75$$

which maps to cache block number (75 modulo 64) = 11.

ACCESSING A CACHE

Notice that this block address is the block containing all addresses between

$$\left[\frac{\text{Byte address}}{\text{Bytes per block}}\right] \times \text{Bytes per block}$$

And $$\left[\frac{\text{Byte address}}{\text{Bytes per block}}\right] \times \text{Bytes per block} + (\text{Bytes per block} - 1)$$

This block maps all addresses between 1200 and 1215.

ACCESSING A CACHE

- Larger blocks exploit spatial locality to lower miss rates.
  - The miss rate may go up eventually if the block size becomes a significant fraction of the cache size, because the number of blocks that can be held in the cache will become small, and there will be a great deal of competition for those blocks. As a result, a block will be bumped out of the cache before many of its words are accessed.

- A more serious problem associated with just increasing the block size is that the cost of a miss increases.
  - Miss penalty is determined by the time required to fetch the block from the next lower level of the hierarchy and load it into the cache. The time to fetch the block has two parts: the latency to the first word and the transfer time for the rest of the block. Unless we change the memory system, the transfer time—and hence the miss penalty—will likely increase as the block size increases.

ACCESSING A CACHE

Design the memory to transfer larger blocks more efficiently, then we can increase the block size and obtain further improvements in cache performance.

- Although it is hard to do anything about the longer latency component of the miss penalty for large blocks, we may be able to hide some of the transfer time so that the miss penalty is effectively smaller.
- The simplest method for doing this, called early restart, is simply to resume execution as soon as the requested word of the block is returned, rather than wait for the entire block.
- This technique is usually less effective for data caches because it is likely that the words will be requested from the block in a less predictable way, and the probability that the processor will need another word from a different cache block before the transfer completes is high. If the processor cannot access the data cache because a transfer is ongoing, then it must stall.

ACCESSING A CACHE

- An even more sophisticated scheme is to organize the memory so that the requested word is transferred from the memory to the cache first. The remainder of the block is then transferred, starting with the address after the requested word and wrapping around to the beginning of the block.

- This technique, called requested word first or critical word first, can be slightly faster than early restart, but it is limited by the same properties that limit early restart.

HANDLING CACHE MISSES

- The control unit must detect a miss and process the miss by fetching the requested data from memory (or, as we shall see, a lower-level cache). If the cache reports a hit, the computer continues using the data as if nothing happened.

- The cache miss handling is done in collaboration with the processor control unit and with a separate controller that initiates the memory access and refills the cache.

- For a cache miss, we can stall the entire processor, essentially freezing the contents of the temporary and programmer-visible registers, while we wait for memory.

HANDLING CACHE MISSES

Let’s look at how instruction misses are handled.

- If an instruction access results in a miss, then the content of the Instruction register is invalid. To get the proper instruction into the cache, we must be able to instruct the lower level in the memory hierarchy to perform a read. Since the program counter is incremented in the first clock cycle of execution, the address of the instruction that generates an instruction cache miss is equal to the value of the program counter minus 4. Once we have the address, we need to instruct the main memory to perform a read. We wait for the memory to respond (since the access will take multiple clock cycles), and then write the words containing the desired instruction into the cache.

HANDLING CACHE MISSES

The steps to be taken on an instruction cache miss:

1. Send the original PC value (current PC – 4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
4. Restart the instruction execution at the first step, which will re-fetch the instruction, this time finding it in the cache.

The control of the cache on a data access is essentially identical: on a miss, we simply stall the processor until the memory responds with the data.

HANDLING WRITES

- Suppose on a store instruction, we wrote the data into only the data cache (without changing main memory); then, after the write into the cache, memory would have a different value from that in the cache. In such a case, the cache and memory are said to be inconsistent.

- The simplest way to keep the main memory and the cache consistent is always to write the data into both the memory and the cache. This scheme is called write-through.

- The other key aspect of writes is what occurs on a write miss. We first fetch the words of the block from memory. After the block is fetched and placed into the cache, we can overwrite the word that caused the miss into the cache block. We also write the word to main memory using the full address.

- Although this design handles writes very simply, it would not provide very good performance.

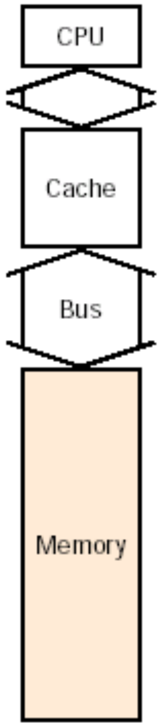
HANDLING WRITES

- With a write-through scheme, every write causes the data to be written to main memory. These writes will take a long time, likely at least 100 processor clock cycles, and could slow down the processor considerably.

- One solution to this problem is to use a write buffer. A write buffer stores the data while it is waiting to be written to memory. After writing the data into the cache and into the write buffer, the processor can continue execution. When a write to main memory completes, the entry in the write buffer is freed. If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.

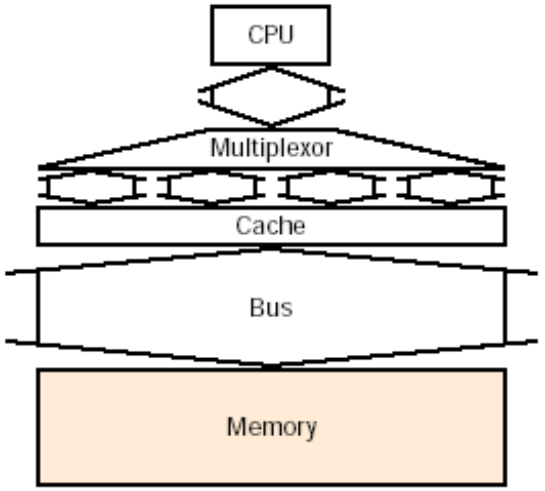
- The alternative to a write-through scheme is a scheme called write-back. In a write-back scheme, when a write occurs, the new value is written only to the block in the cache. The modified block is written to the lower level of the hierarchy when it is replaced. Write-back schemes can improve performance, especially when processors can generate writes as fast or faster than the writes can be handled by main memory.

DESIGNING MEMORY SYSTEM TO SUPPORT CACHE



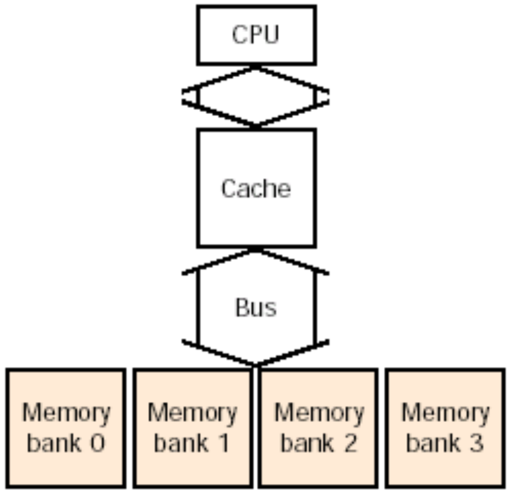
a. One-word-wide memory organization

all components are one word wide



b. Wide memory organization

a wider memory, bus and cache are utilized



c. Interleaved memory organization

interleaved memory banks with a narrow bus and cache are utilized

DESIGNING MEMORY SYSTEM TO SUPPORT CACHE

Assume that it takes

- 1 clock cycle to send the referenced address
- 15 clock cycles for each DRAM access initiated
- 1 clock cycle to send a word of data

If cache block is four words and

a. one word wide bank of DRAM,

Miss penalty = 1 + (4 x 15) + (4 x 1) = 65 clock cycle

Number of byte transferred per clock cycle for a single miss $\frac{4 \times 4}{65} = 0.25$ bytes per clock cycle

b. wider memory, bus and cache.

With a main memory width of 2 words, Miss penalty = 1 + (2 x 15) + (2 x 1) = 33 clock cycle, and number of bytes transferred $\frac{4 \times 4}{33} = 0.48$ bytes per clock cycle.

- Wider bus and higher cache access time

DESIGNING MEMORY SYSTEM TO SUPPORT CACHE

Assume that it takes

- 1 clock cycle to send the referenced address
- 15 clock cycles for each DRAM access initiated
- 1 clock cycle to send a word of data

c. Interleaved memory banks with a narrow bus and cache.

With four banks, the time to get a four-word block would consist of 1 cycle to transmit the address and read request to the banks, 15 cycles for all four banks to access memory, and 4 cycles to send the four words back to cache.

Miss penalty = 1 + (1x15) + (4 x 1) = 20 clock cycles, and number of bytes transferred $\frac{4\times4}{20} = 0.80$ bytes per clock cycle