Memory Read and write operations

Read (using Multiplexers):

1. Transfer the binary address of the desired word to the address line (tells the memory where to read from)
2. Activate read control line (tells memory to perform read operation)

Write (using decoder):

1. Transfer the binary address of the desired word to the address lines (tells the memory where to store the data)
2. Transfer data bits that must be stored in memory to the data output line (data we want to write)
3. Activate the write control line (tells memory to perform write operation)

Temporal Locality: When an instruction is executed or data is accessed, it is stored in the cache because there is a high probability it will be accessed again.

eg. (loop variables)

```
while (condition) {
    i++; // access variable
} // for 100 iterations, the variable would be referenced again and again
```

Spatial Locality: When an instruction is executed or data is accessed, nearby items are also loaded into the cache because there's a high probability they'll be accessed soon

eg. Arrays and vectors

Cache Hit: a memory access where the data is already in cache
Cache Miss: a memory access where data isn't in the cache
Hit ratio: (#of cache accesses)/(# of total accesses)

eg.

× ✓ ✓ ✓ × × ✓ ✓

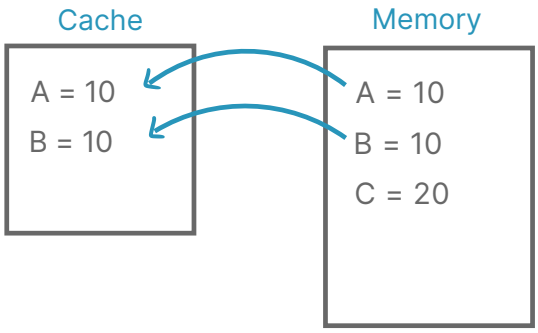
Data to access → 5, 5, 5, 5, 6, 7, 6, 5

Hit ratio: 5/8

× Cache Miss
✓ Cache Hit

Data gets into the cache by a read operation only.

Cache/Memory Example: C = A + B (read A, read B, write C)



What we bring in-to the cache is based on the principle of locality.

Direct Mapping: Each block of main memory maps to only one cache line: (block #) mod (# of lines)

Direct mapped cache example:

Cache size 4 (r = 2) , memory size: 32 blocks (s = 5)
Line 0: block 0, 4, 8, 12
Line 1: block 1, 5, 9, 13

Replacement Policies:

- When cache is full, a line must be replaced
- Most common strategy: Least Recently Used (LRU)

Write Policies:

- write-through: Update RAM every time cache is updated
- Write-back: Delay RAM update until block is evicted from cache