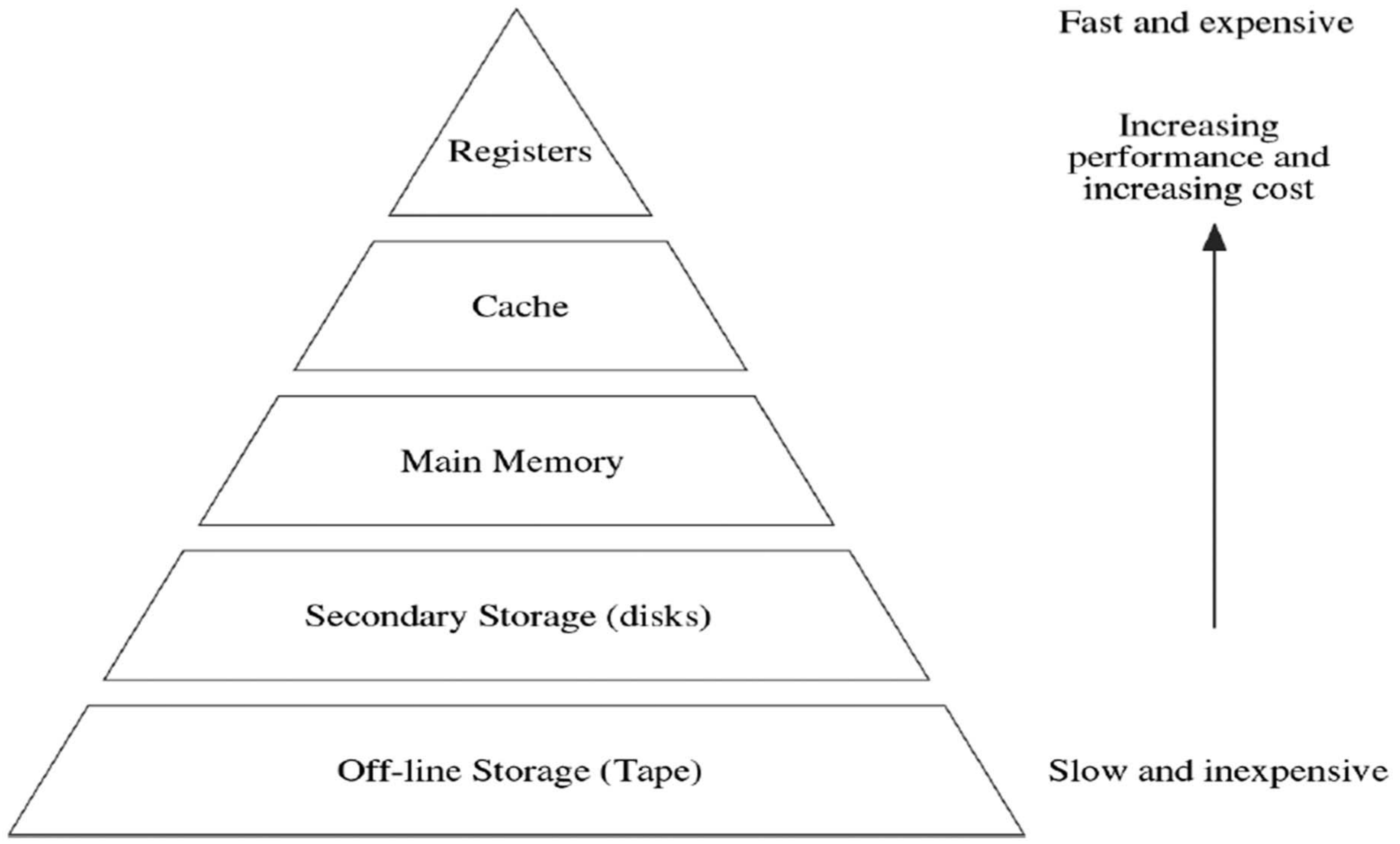
Memory Sub-Systems

- Memory Hierarchy
- Principle of Locality
- Access Times
- Memory Map

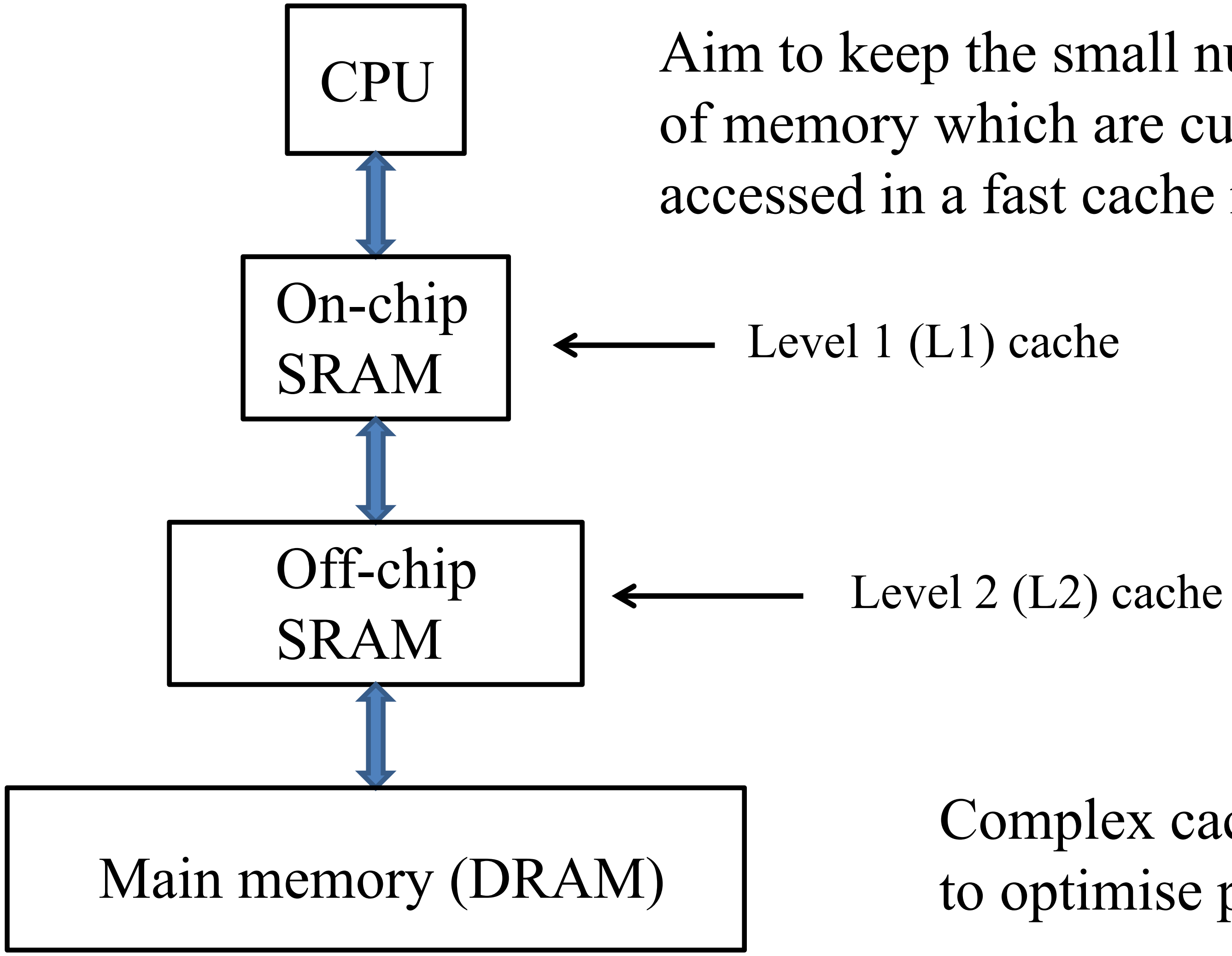

Registers
Cache
Main Memory
Secondary Storage (disks)
Off-line Storage (Tape)
Fast and expensive
Increasing performance and increasing cost
Slow and inexpensive

Principle of Locality

- This is really an observation, not a principle.
- Temporal Locality.
    - If a memory location has been accessed, it will *tend* to be accessed again soon
- Spatial Locality.
    - If a given memory location has been accessed, the next location to be accessed will *tend* to be nearby

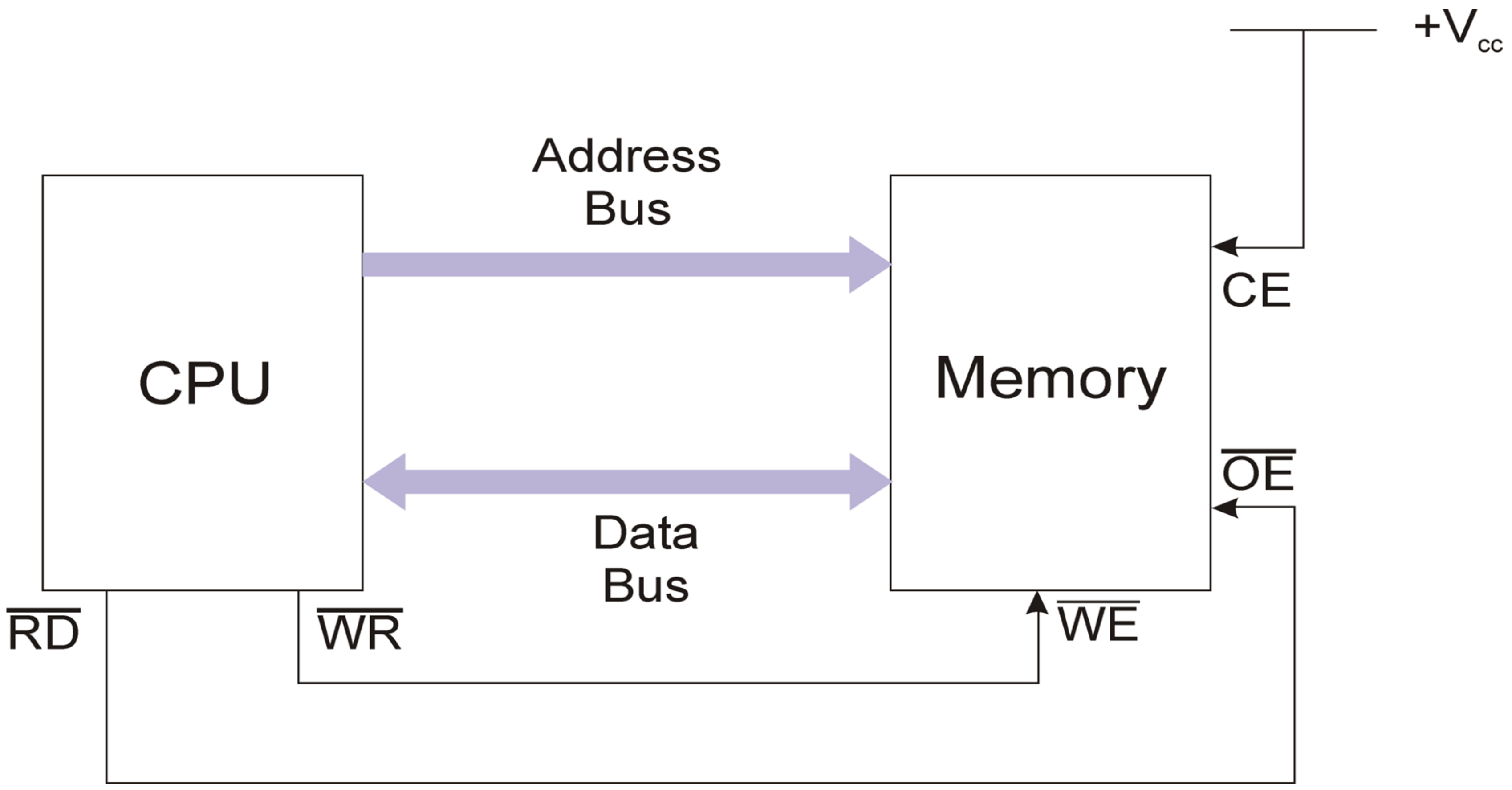
Memory cache schemes aim to exploit both temporal and spatial locality.

Aim to keep the small number of blocks of memory which are currently being accessed in a fast cache memory



Complex cache schemes exist to optimise performance.

CPU-Memory Interfacing

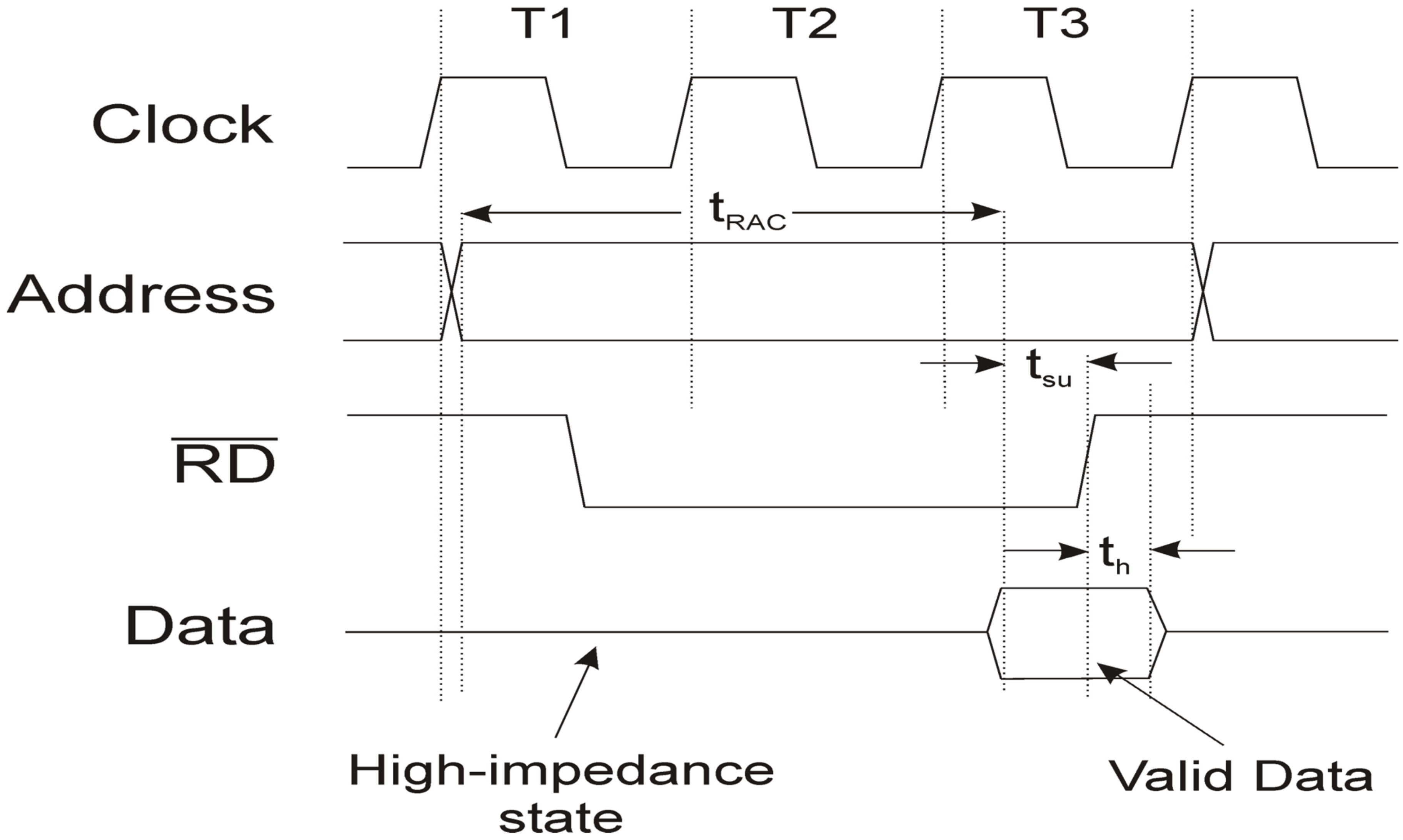


Memory Cycles

The timing behaviour of a memory cycle is normally presented in a timing diagram where timings of signals are related back to the clock.

- For outputs: the timings are a guarantee of what the μP will do.
- For inputs: the timings are a specification to which the external system MUST adhere.
- All timing-relationships are relevant and during a design, the μP timing information must be related to timing information for devices to which it will be connected.

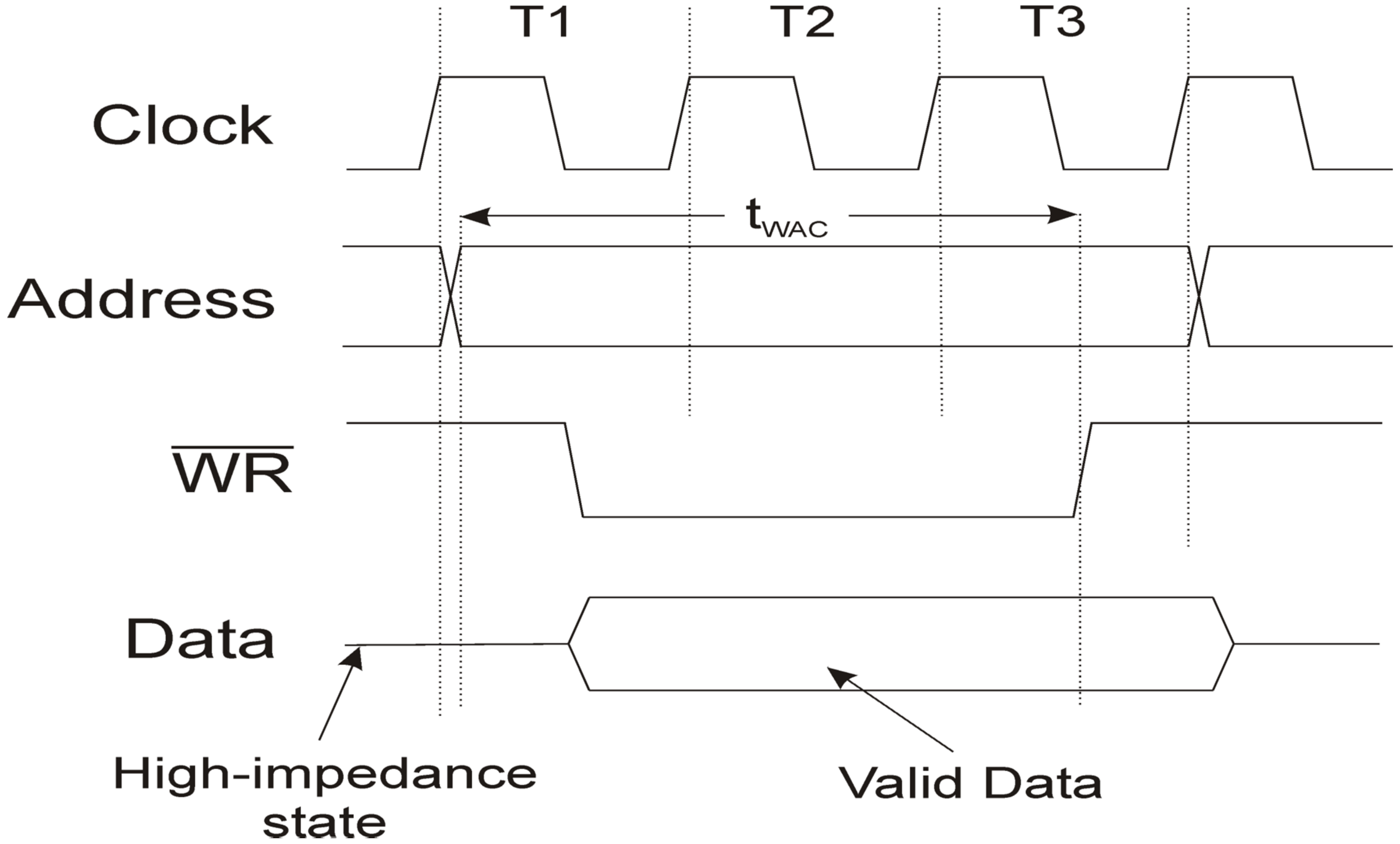
Typical Memory Read



if read

1 Set up the address of the location to be accessed and enable the memory,
2 Wait,
3 Set up a signal indicating that data is to be retrieved **($\overline{\mathbf{RD}}$)**,
4 Wait for the data to be retrieved from the memory and placed on the data bus inputs,
5 Sample the data on the data bus inputs,
6 Disable the memory.

Typical Memory Write

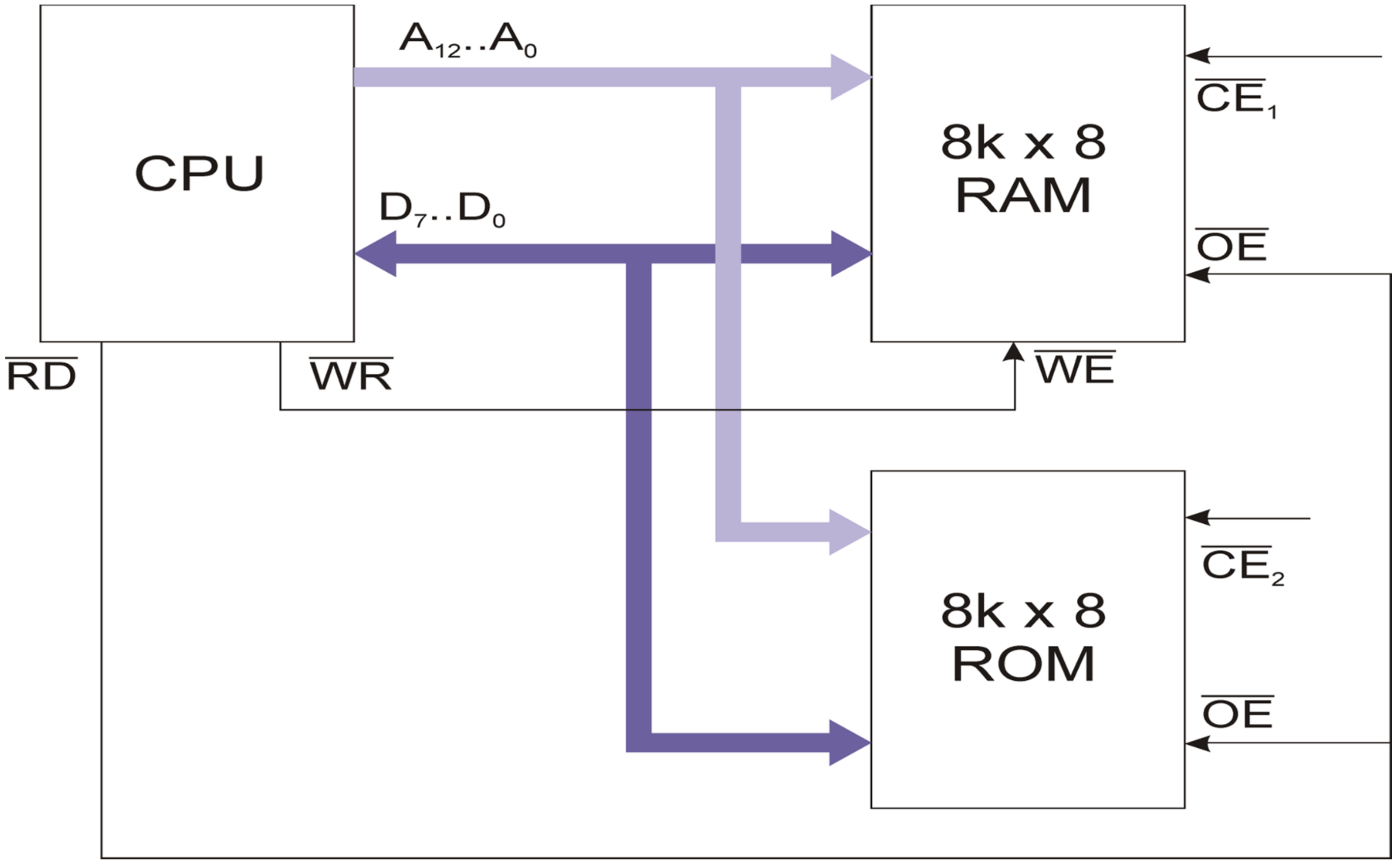


if write

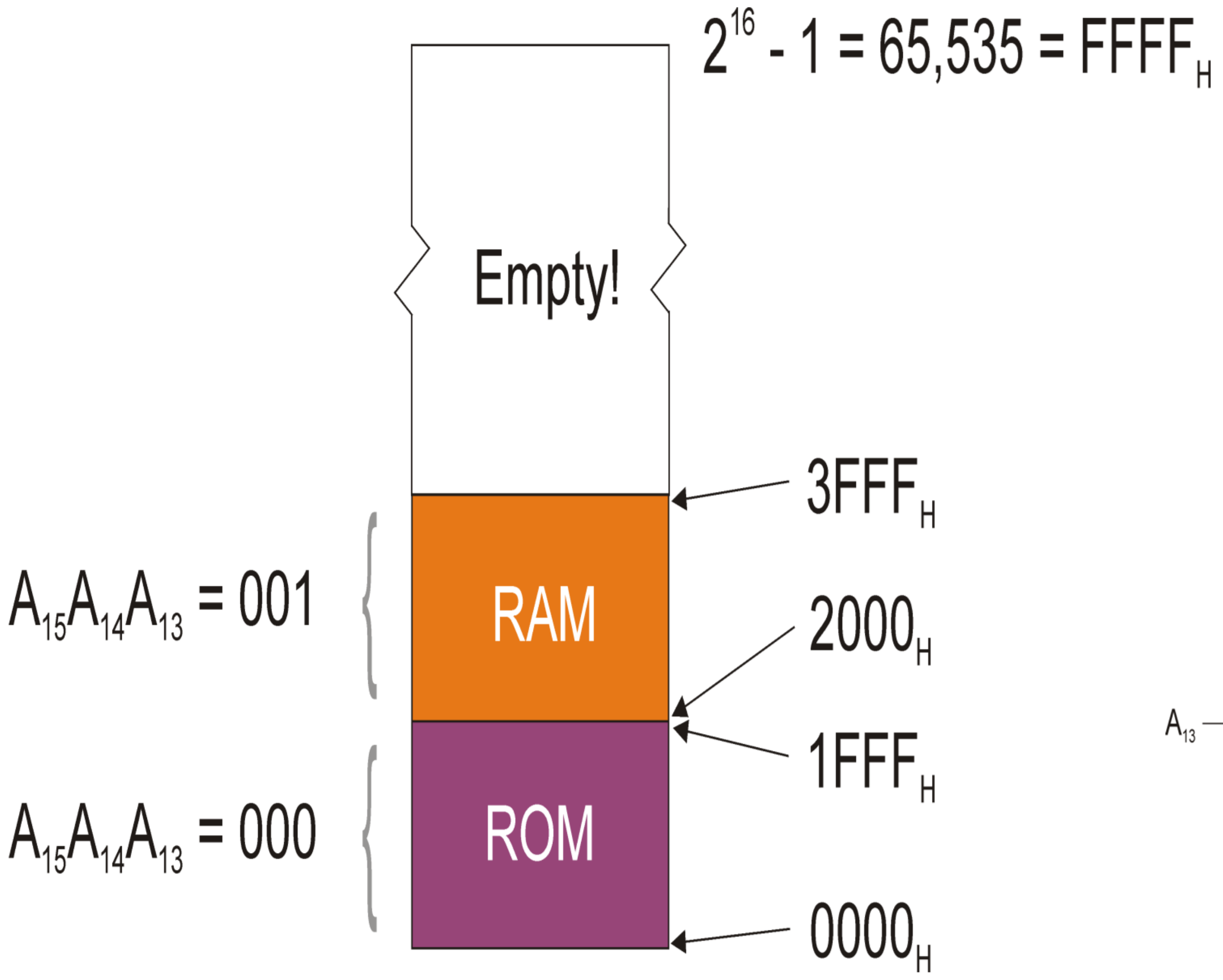
1 Set up the address of the location to be accessed and enable the memory,
2 Wait,
3 Set up a signal indicating that data is to be stored ($\overline{\text{WR}}$),
4 Place the data to be written on the data bus outputs (which must be connected to the memory data inputs,
5 Wait for the memory device to store (or prepare to store) the data,
6 Signal the end of the write and disable the memory.

Memory interfacing to two memory chips

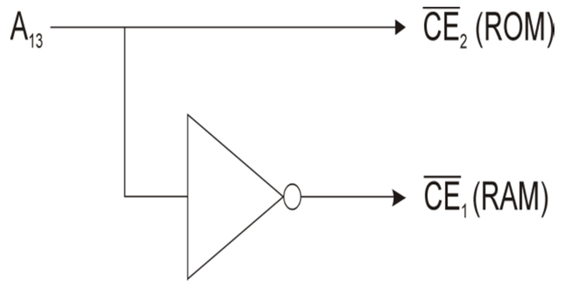
CPU has 8 bit data bus
and 16 bit address bus.



Memory map

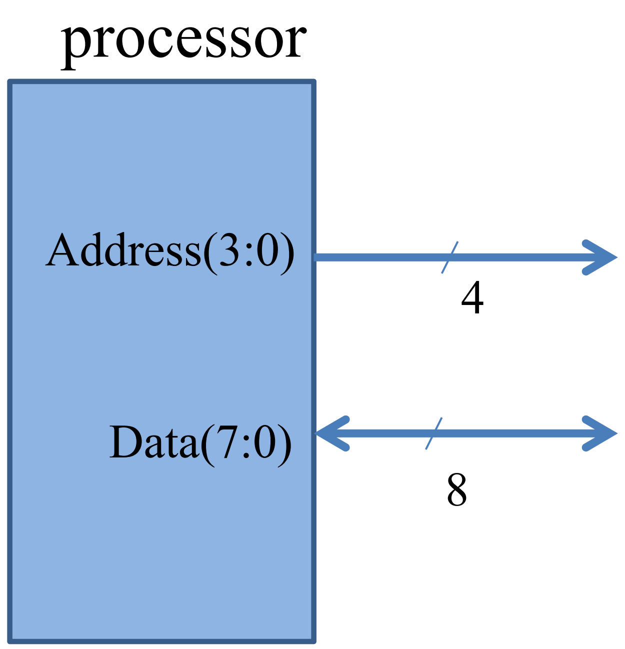


$2^{16} - 1 = 65{,}535 = FFFF_H$
Empty!
$3FFF_H$
$A_{15}A_{14}A_{13} = 001$
RAM
2000_H
$1FFF_H$
$A_{15}A_{14}A_{13} = 000$
ROM
0000_H



A_{13}
$\overline{CE_2}$ (ROM)
$\overline{CE_1}$ (RAM)

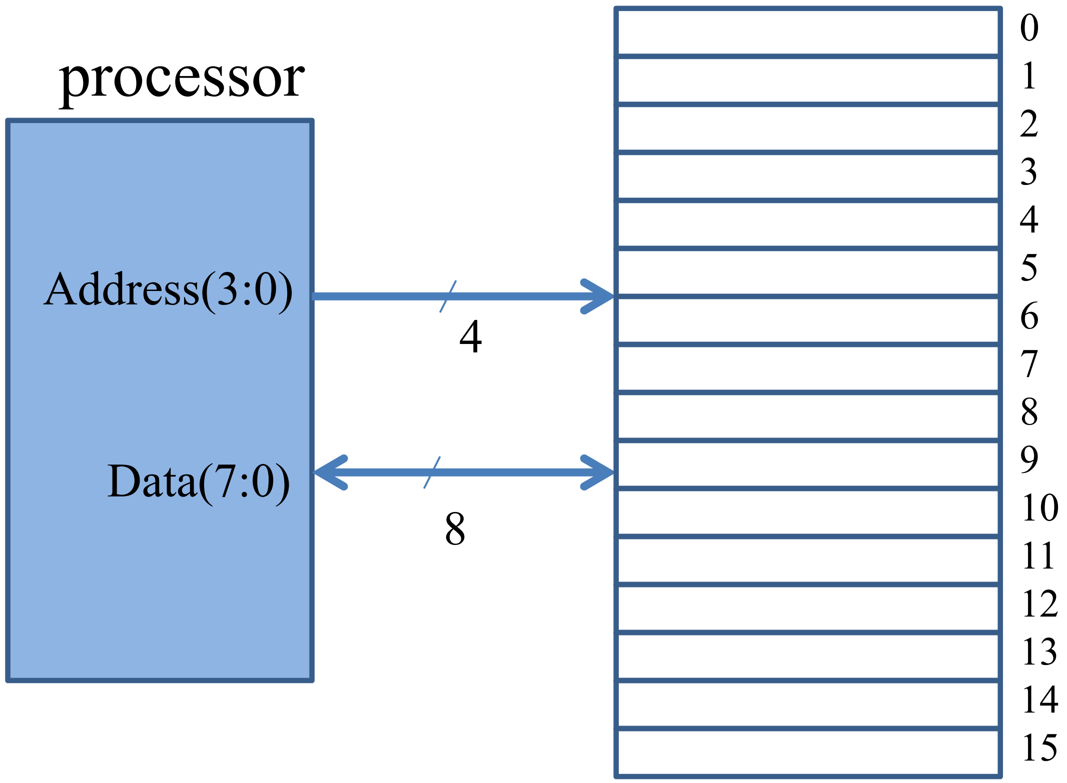
Memory interfacing to two memory chips
Example 2:



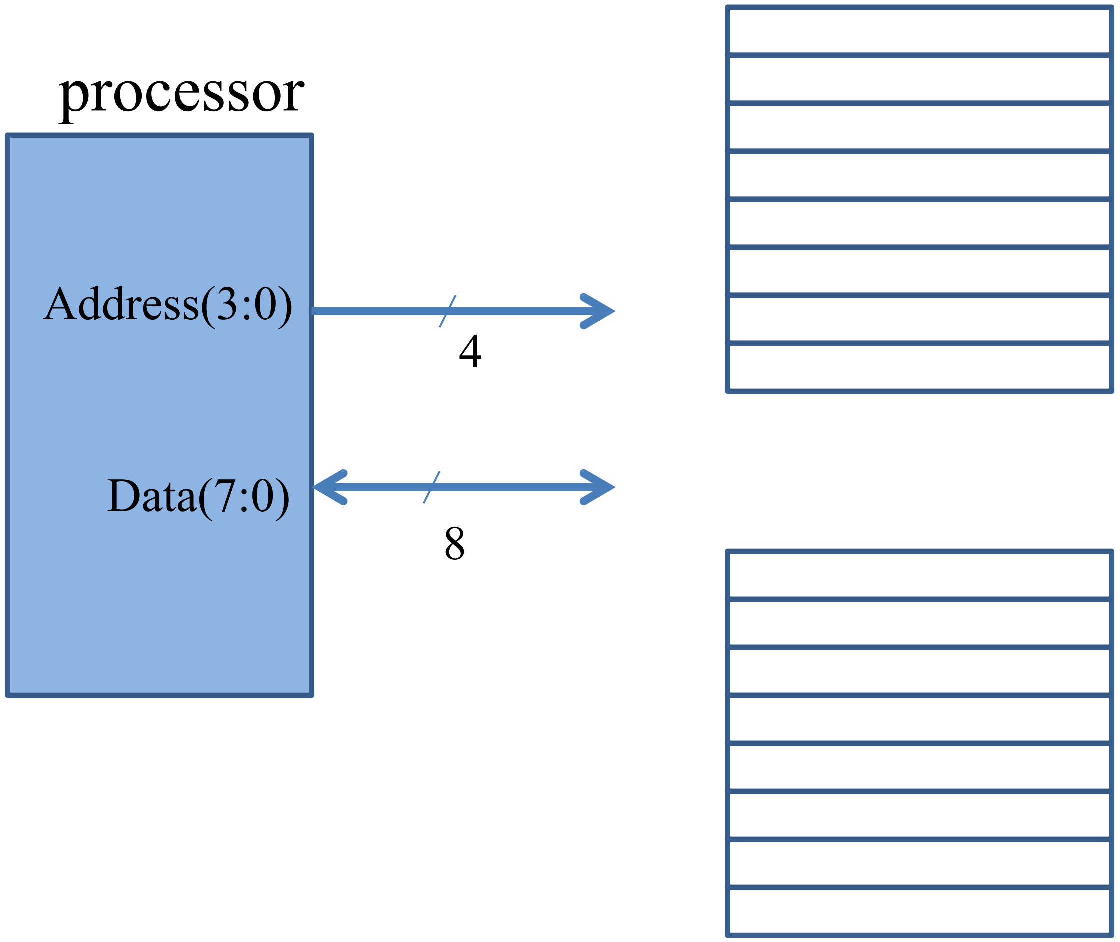
4 address bits gives 16 addressable locations

8 bit data bus can write a byte of data to the memory

single memory with 16 addressable locations

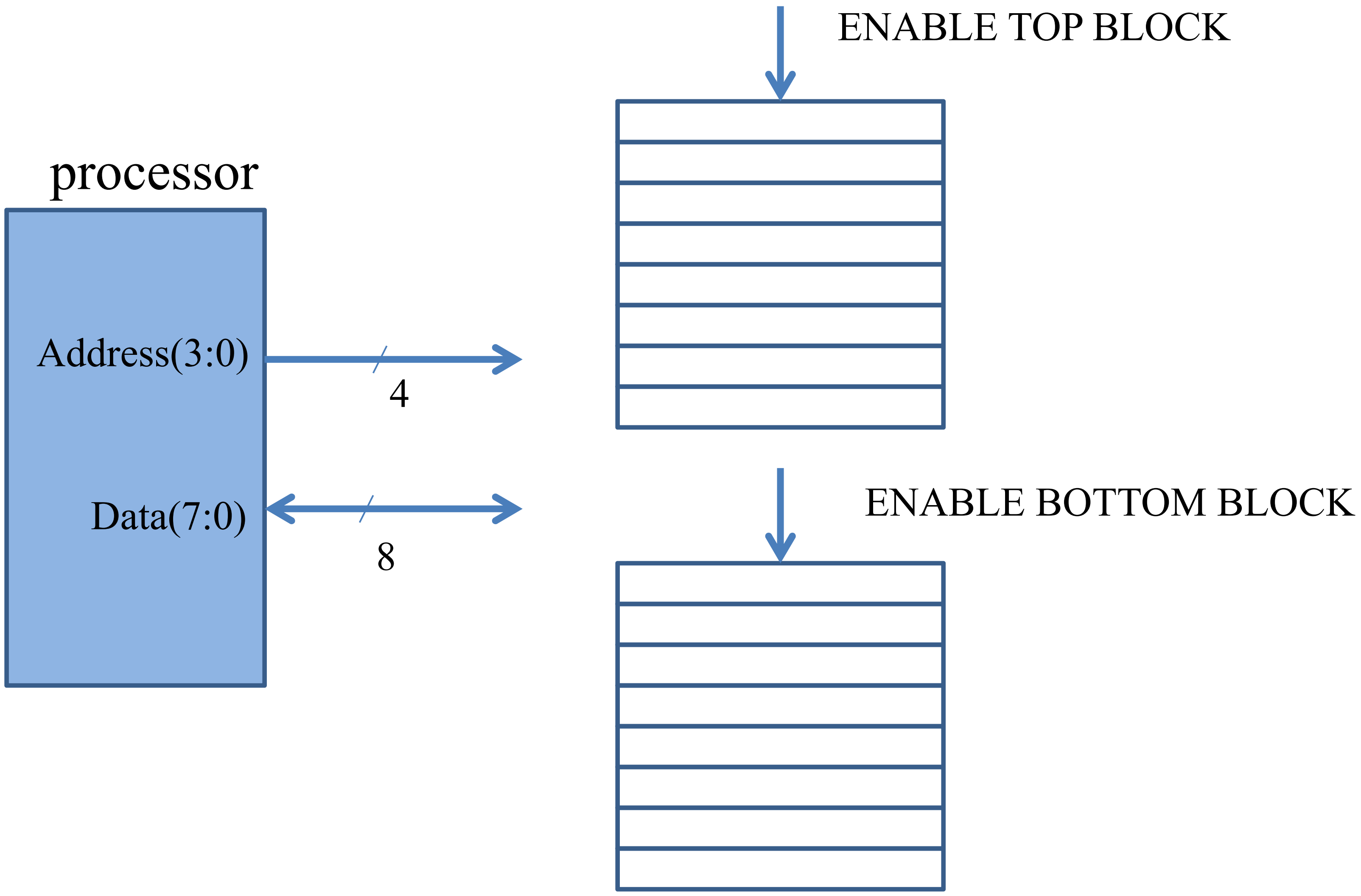


the single 16 row memory could be replaced by two 8 row memory chips

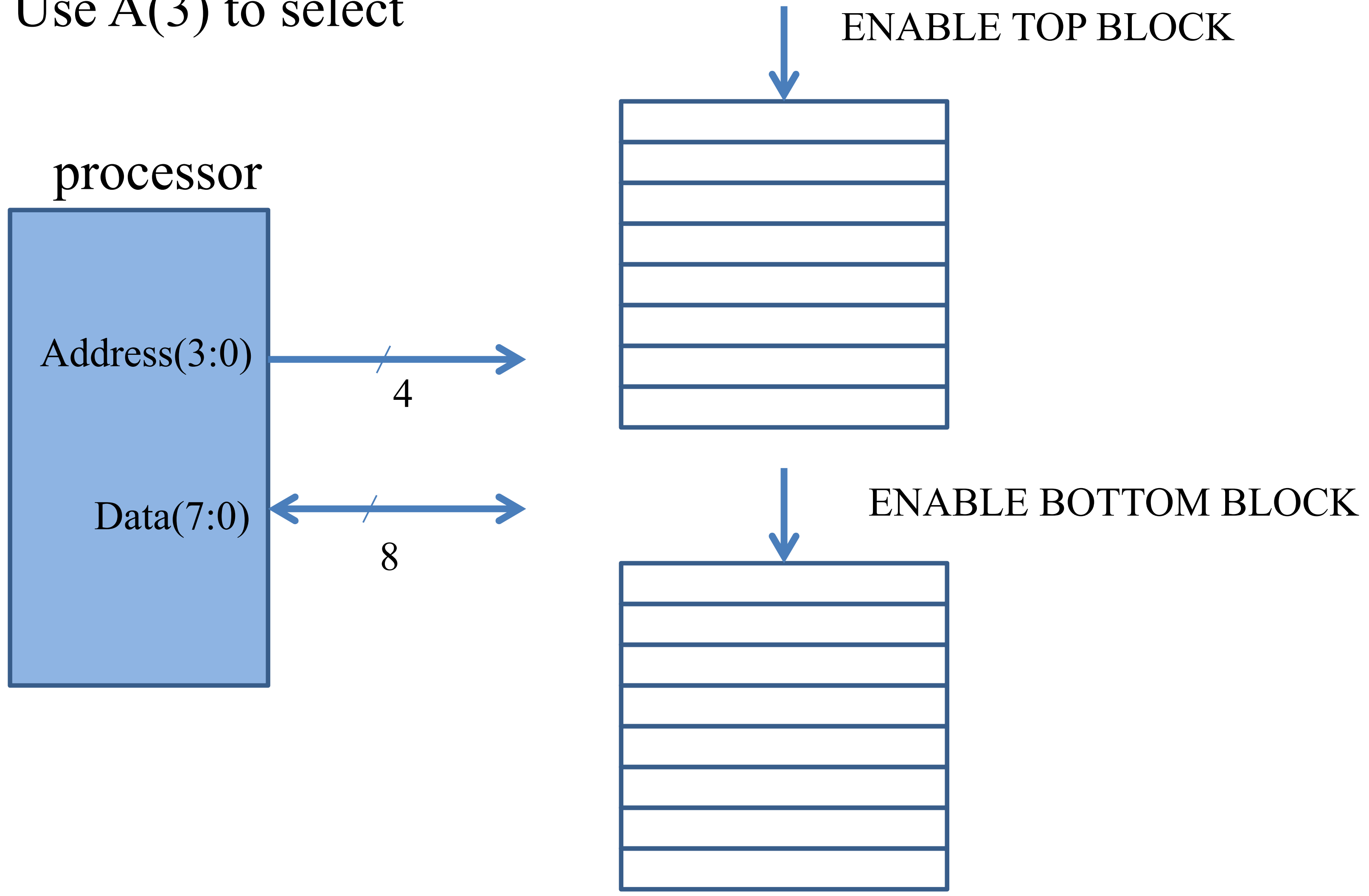


0	0 000
1	0 001
2	0 010
3	0 011
4	0 100
5	0 101
6	0 110
7	0 111
8	1 000
9	1 001
10	1 010
11	1 011
12	1 100
13	1 101
14	1 110
15	1 111

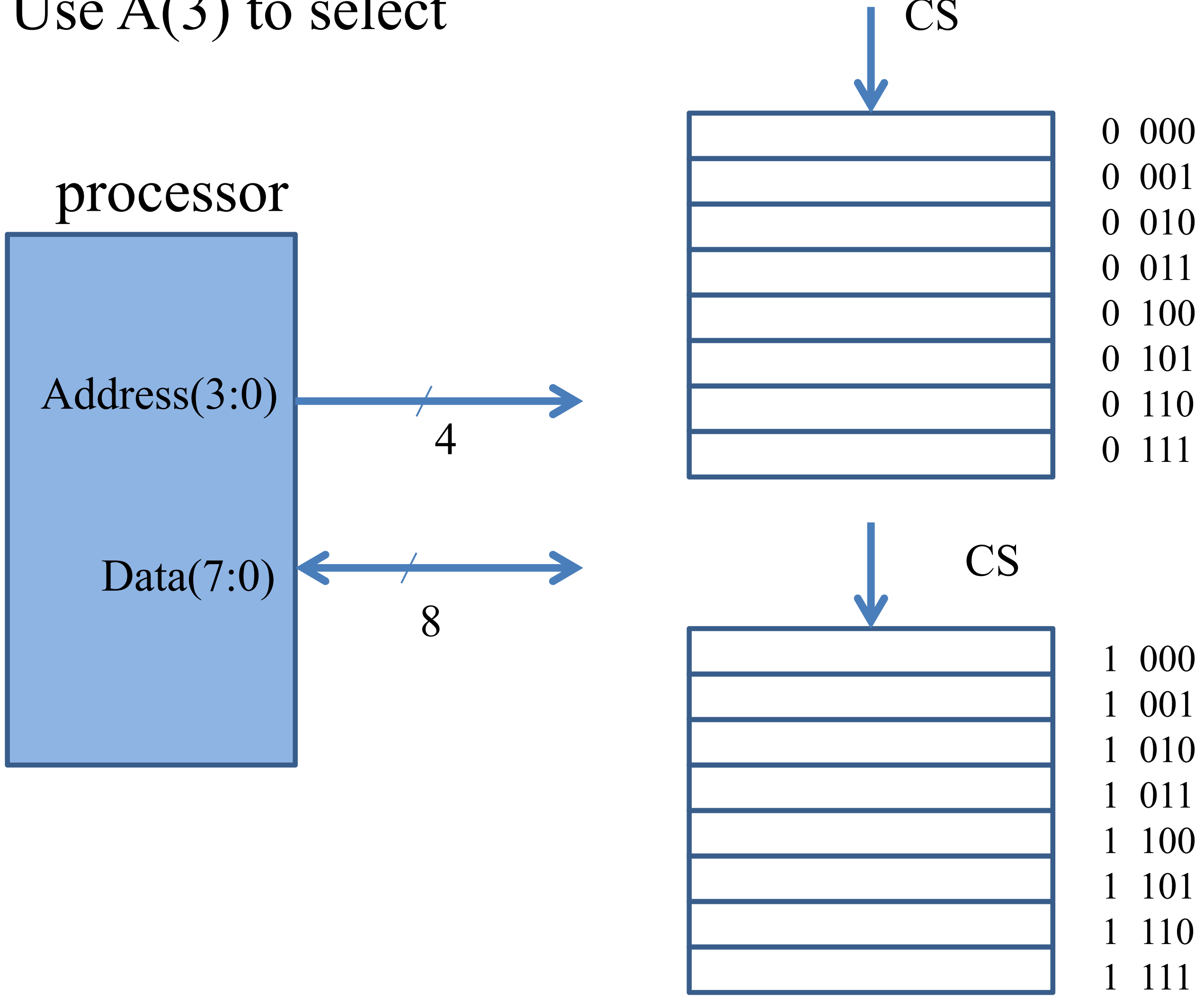
note how the msb changes in the count sequence


ENABLE TOP BLOCK
processor
Address(3:0)
4
Data(7:0)
8
ENABLE BOTTOM BLOCK

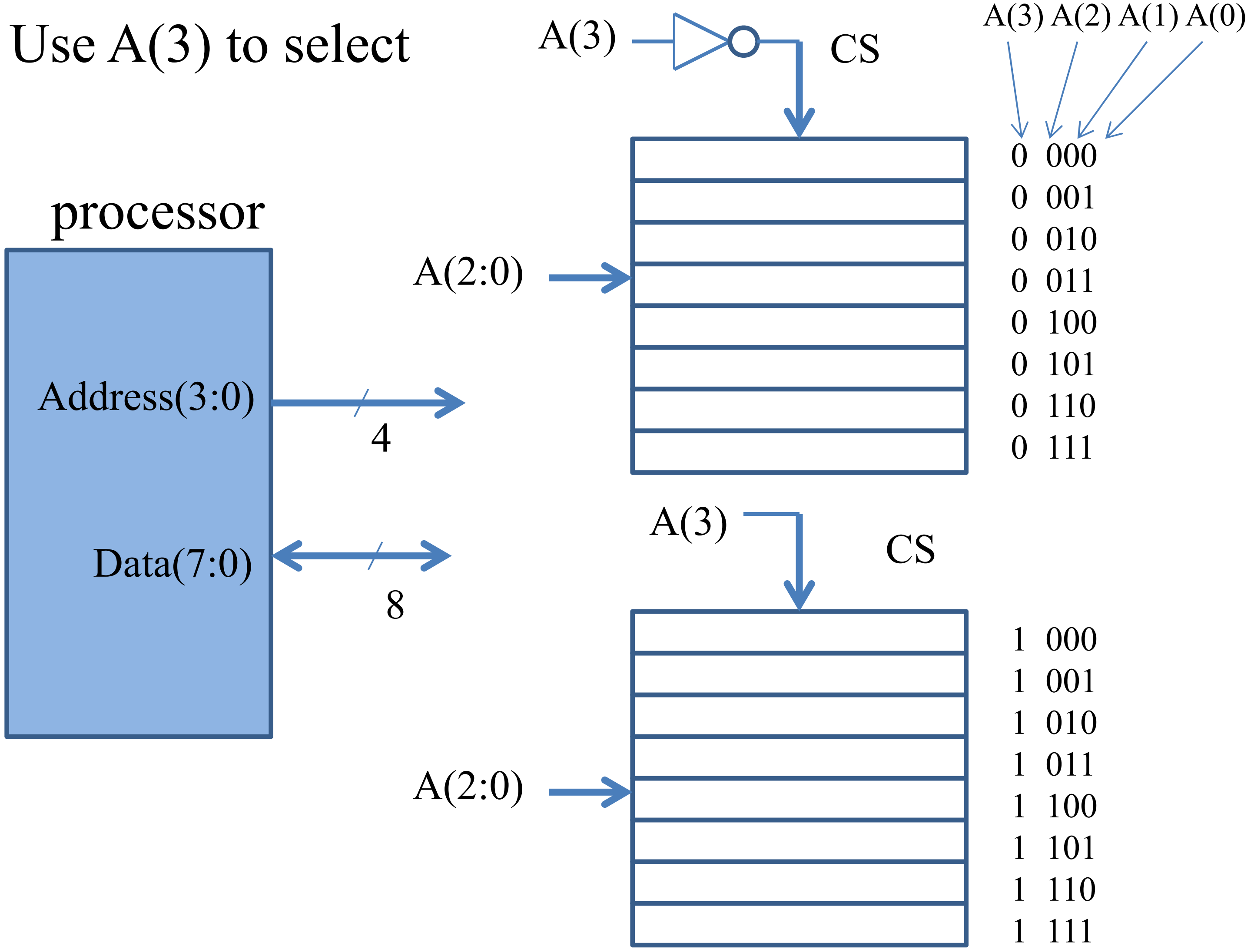
Use A(3) to select



Use A(3) to select



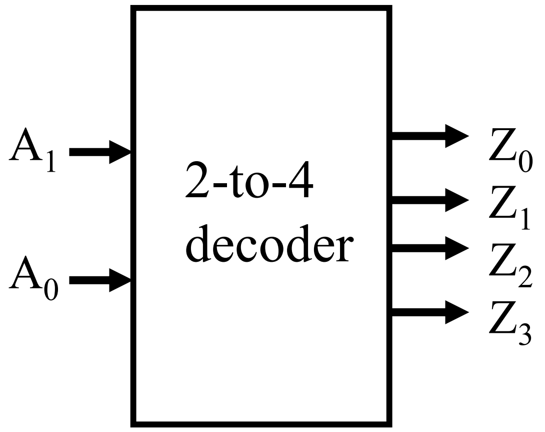
Use A(3) to select



However, the device is unlikely to occupy the whole address space. How is the device placed into the address map?

- The least significant address bits are routed to the address inputs, the most significant are routed to an 'address decoder' which generates the CS input.
- If CS is inactive the device ignores all their inputs and keeps its outputs inactive.
- 'Decoding' the address corresponds to driving CS active only when a certain pattern of bits appears on the decoded address bits.

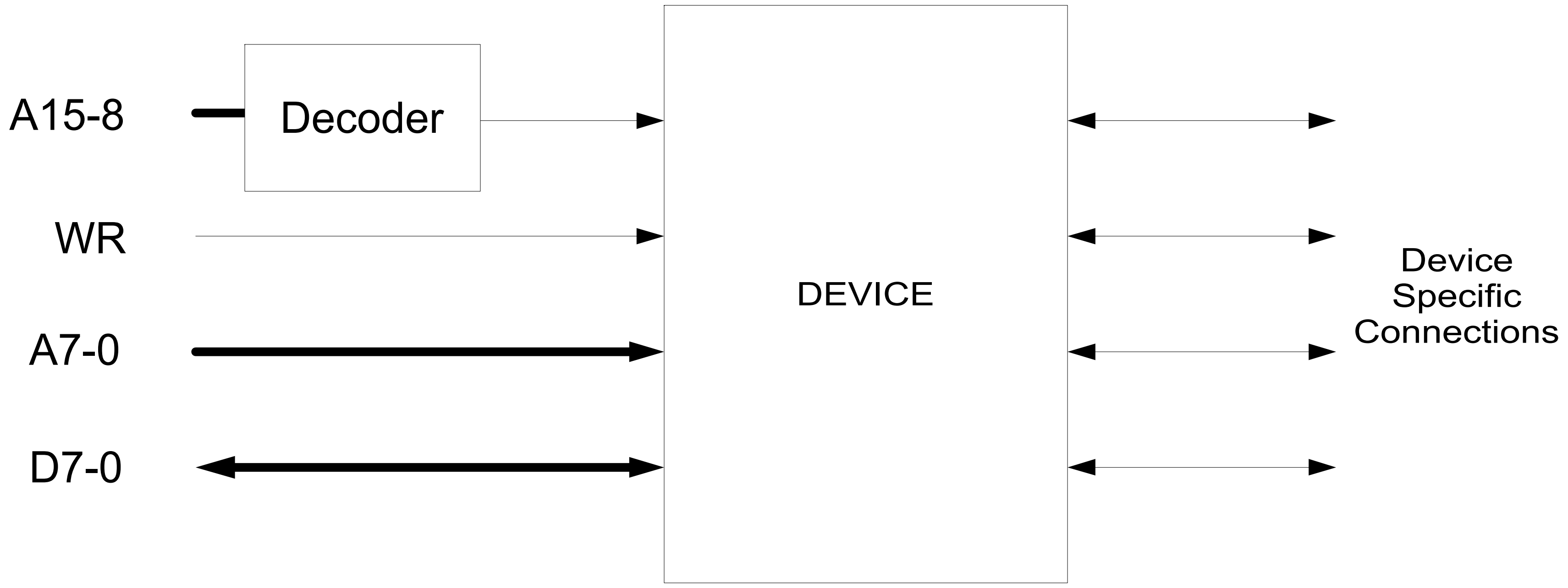
Decoders



A_1	A_0	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

An n-input decoder has 2^n output lines.

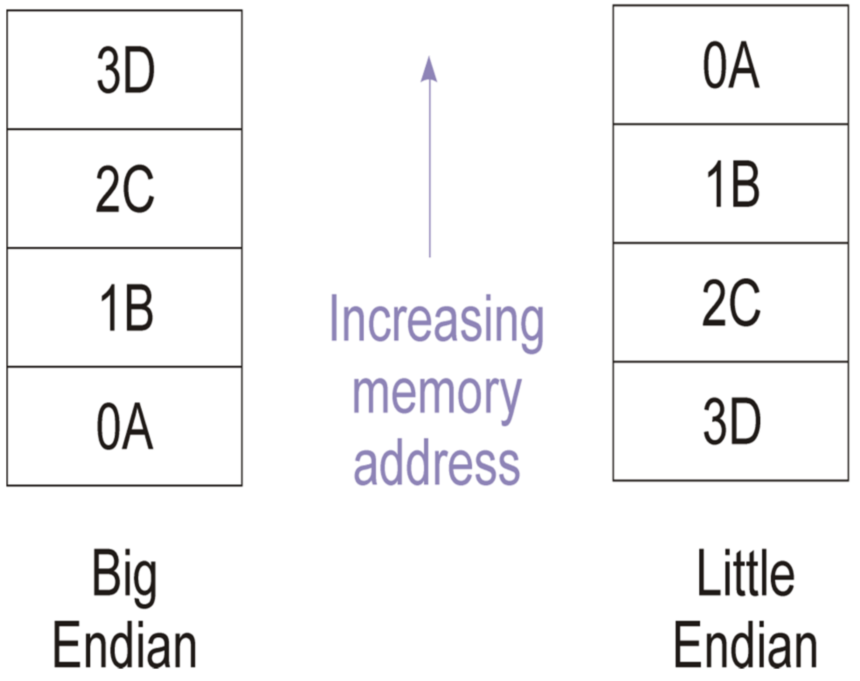
- In this case, A(7:0) are inputs to the device (256 internal locations). A(15:8) are decoded, say at 83_H, by the decoding logic.
- The device will be activated at any address beginning 83_H. This is the range 8300_H to $83FF_H$.



- Devices usually occupy address ranges which are a power of 2, and are based at addresses which are a multiple of this power of 2. This simplifies the address decoding circuitry.
- Furthermore, sometimes all of the address bits are not decoded. This simplifies decoding even more but causes the block of memory to be 'mirrored', i.e. to appear at other addresses in the memory map.
- So, if only A15..12 were decoded at 8_H, the device would appear between 8000_H to $80FF_H$, 8100_H to $81FF_H$ .. $8F00_H$ to $8FFF_H$.
- Different devices can occupy different ranges of addresses. The only constraint is that *no two address ranges should overlap*.

Endian-ness

Considering storing the data OA1B2C3D$_H$ in a byte wide memory. There are two possibilities.


3D
2C
1B
0A
Big Endian
Increasing memory address
0A
1B
2C
3D
Little Endian