Memory Systems Design

- Introduction
  - Classification of Memory Technology
  - Memory and CPU Interfacing
- Address Decoding
  - Full Address Decoding Techniques
  - Partial Address Decoding Techniques
  - Block Address Decoding Techniques
  - Address Decoding using GALs

Memory Systems

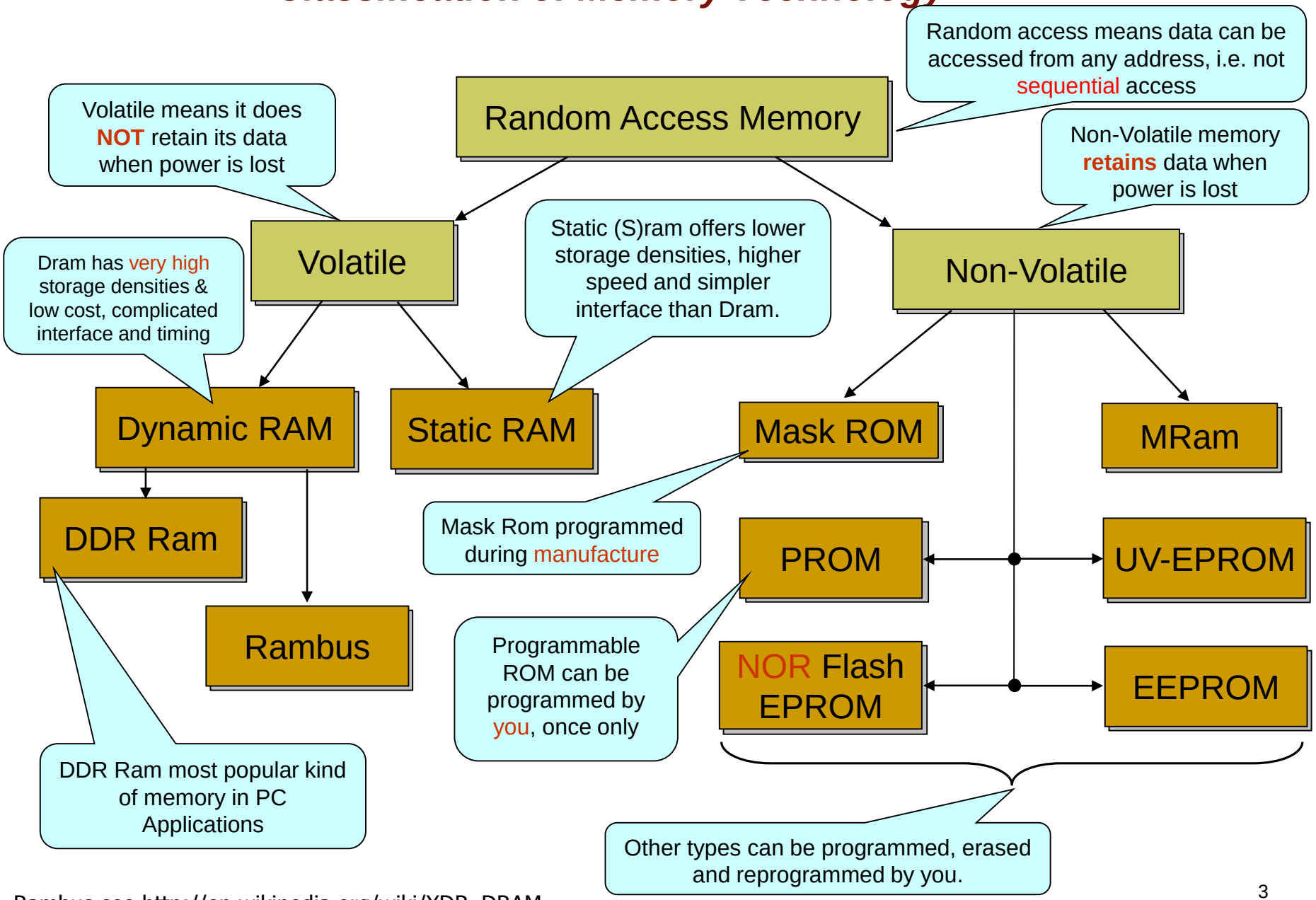
Introduction

- Memory systems form an integral part of all microcomputer systems and are used to store the following:-

  - Program Code – The instructions in your Java or C/C++ code.
  - Data – For the variables and stack you create into your program
  - Firmware – i.e. configuration data and Bios programs, i.e. information or programs that you wish to retain after the system has been switched off and which will be available instantly when the power is reapplied.

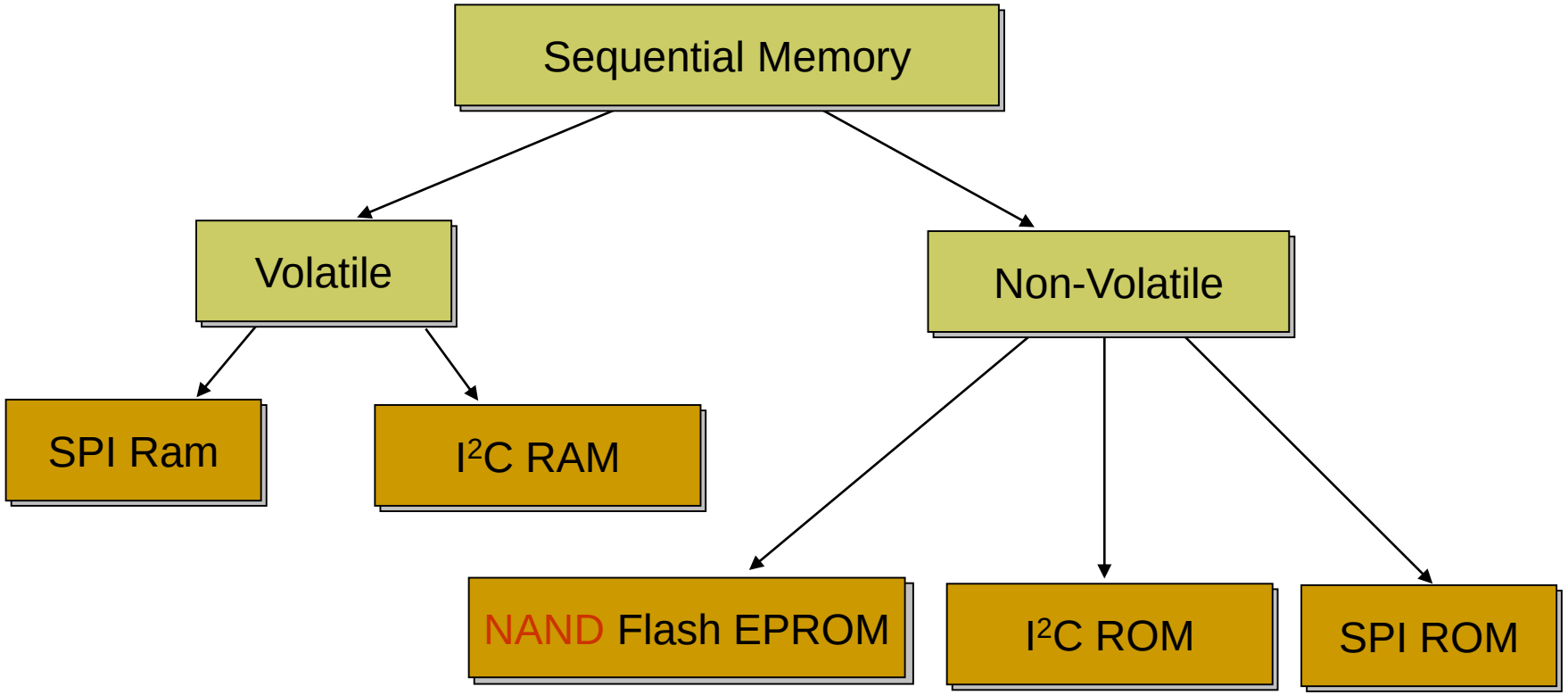
- Most computer systems thus need a variety of different memory technologies to make a working system and that memory technology can be classified in a variety of ways as shown in the next slide, based on speed, density, cost, size of packaging, ease of use and volatility.

Classification of Memory Technology



Rambus see http://en.wikipedia.org/wiki/XDR_DRAM

Classification of Memory Technology

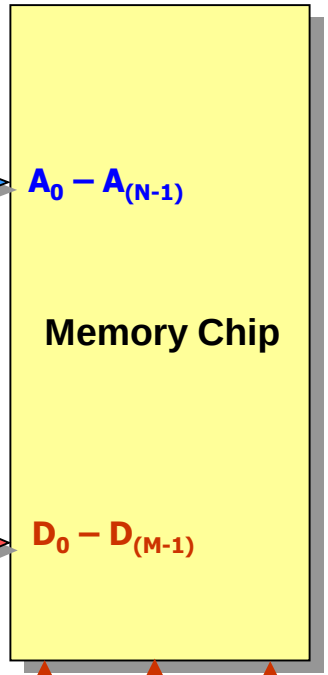


Sequential devices are non-randomly accessible and are characterised by different types of external interface: An access is made by first writing an address to them, they respond with data from successive locations with each subsequent read/write access:

- Low speed (CPU cannot execute programs from them directly)
- Tiny packaging with few external pins (relative to random access devices)
- Protocol Intensive: need a program to read/write to them: useful as secondary or backup storage

Typical Random Access Memory Chip - External Interface

Address Bus – A collection of 'N' address lines A_0 - A_{N-1} used to select 1 of 2^N separate memory locations within the chip - comes from CPU.



Data Bus – A collection of 'M' data lines D_0 - D_{M-1} used to transfer data to/from chip/cpu.

Data Bus will be bi-directional if the device supports reading ***and*** writing. Tri-state outputs when **not** selected or when writing.

'N'	Locations	Address(0 to)
1	2	00000001
2	4	00000003
8	256	000000FF
10	1024 (1k)	000003FF
13	8192 (8k)	00001FFF
16	65536 (64k)	0000FFFF
20	1Meg	000FFFFF
24	16Meg	00FFFFFF
32	4Gig	FFFFFFFF

Total Storage = M * 2^N bits

$\overline{CS}$ – Chip select (active low in this case). There may be more than 1 chip select signal present on the chip in which case all must be active for device to respond.

R/$\overline{W}$ – Read/Write (*if present*). Controls data transfer direction.

$\overline{CS}$ R/$\overline{W}$ $\overline{OE}$

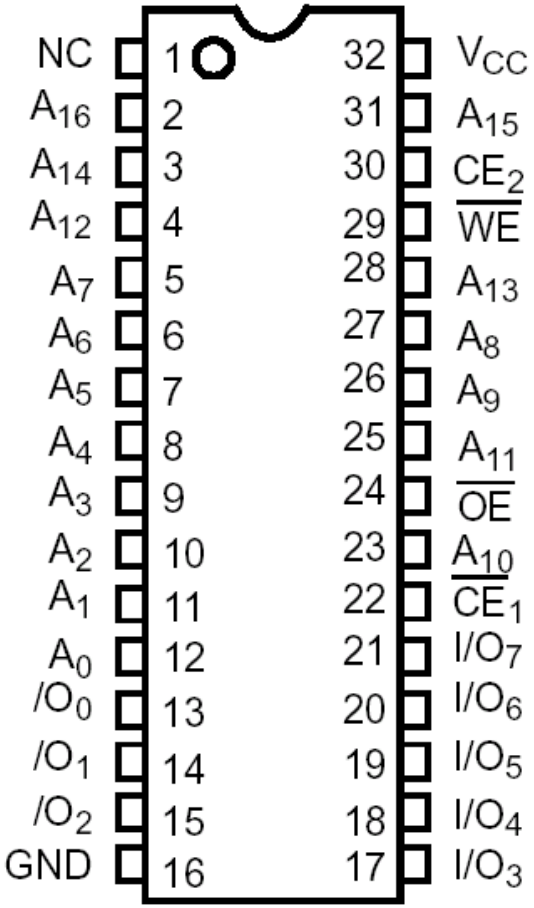
$\overline{OE}$ – Output Enable (If present). Controls tri-state outputs on Data bus. Used to turn off the data bus outputs between memory accesses in fast systems.

Can be tied to $\overline{CS}$ or logic 0 if required.

A Typical Random Access Memory Chip made by Cypress

Question 1

- How many individually addressable memory locations does this device have and what is its range of addresses?



Question 2

- What is the total bit storage capacity of this device?

Question 3

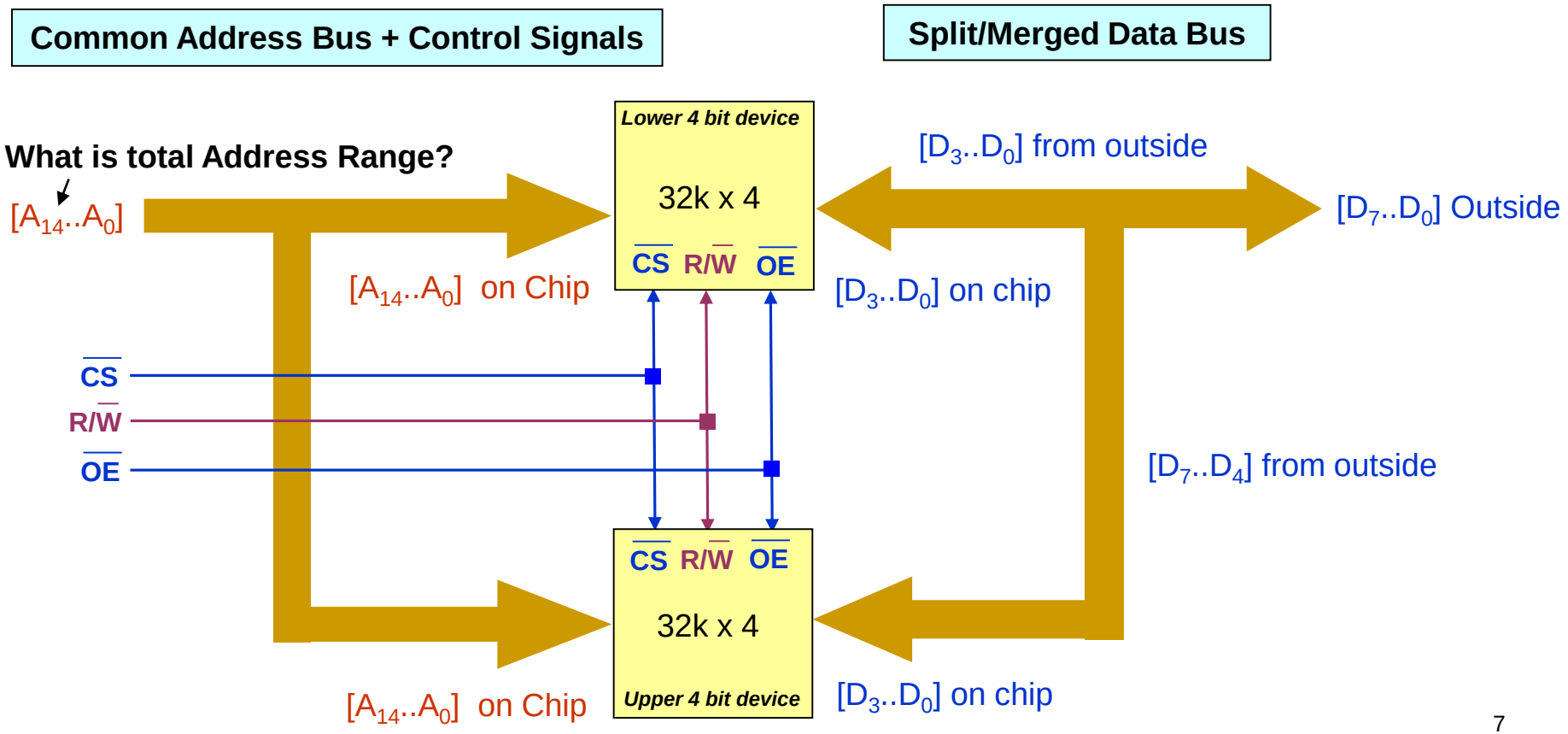
- How many chip select and output enable lines does this chip have and what is their active state?

Question 4

- What does $\overline{WE}$ mean

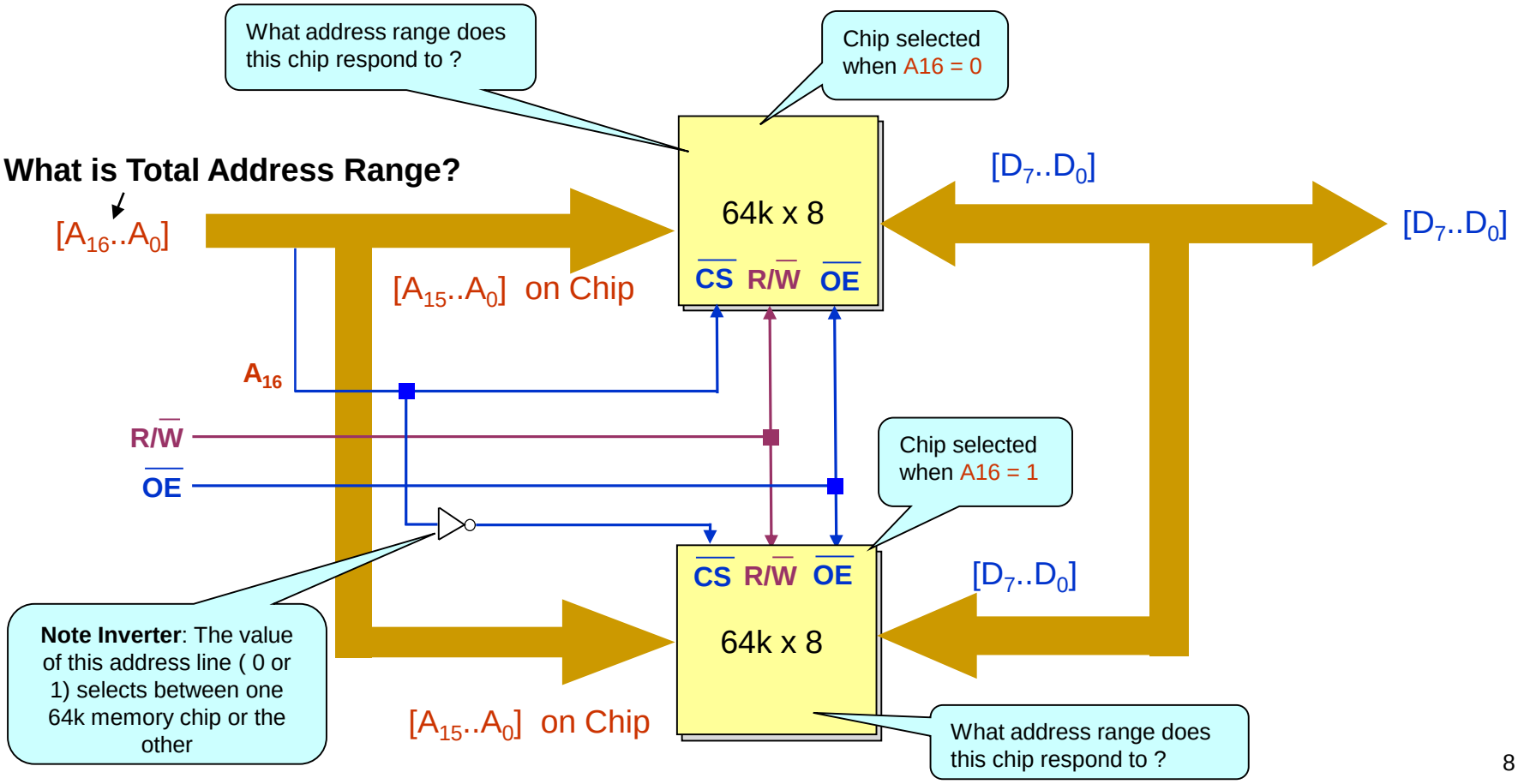
Building Memory Systems by Increasing Data Bus Width

- Sometimes you want a memory system with a certain “data bus width” but you can’t buy chips with that many data pins. For example, how could we make a 32k x 8 bit memory system (byte wide) from 32k x 4 bit memory chips i.e. chips with 4 data lines?
- Put 2 devices to work together in parallel, each dealing with 4 data bits. Each chip is connected to 15 address lines (2^{15} = 32k) labelled A_{14}-A_0



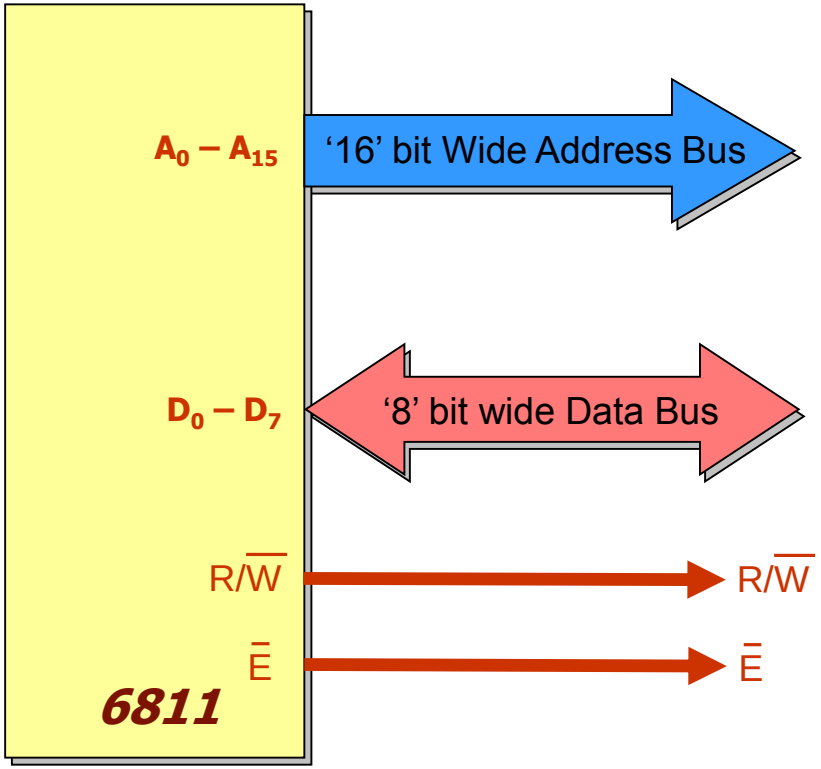
Increasing Addressing Range

- How could we make 128k bytes of memory using 64k byte (64k x 8 bit) chips so that there are no gaps in the address range, i.e. the memory looks like a contiguous block.
- Each 64k byte chip will use 16 address lines, $A_{15} - A_0$ since $(2^{16} = 64k)$.
- A 128k byte block requires 17 address lines $A_{16} - A_0$ since $(2^{17} = 128k)$.
- **Solution**: Use the single unused address line (A_{16}) to select between one or other chip.



Typical Simple 8-bit CPU Interface: Motorola 6811

How are these signals affected by a program?



Address Bus – 16 bits = 64k address space [0000 – FFFF]. An address is issued by the CPU whenever it wishes to access an external memory location (off chip)

Total address space composed of external as well as internal (built in memory and IO) space.

Data Bus – Driven by the CPU when it wishes to write to a memory location. Read by CPU when it wishes to read the contents of memory location

R/$\overline{W}$ – Controls direction of data transfer
Logic 1 means CPU is reading from Memory.
Logic 0 means CPU is writing to memory

$\overline{E}$ (Enable) low when address is Valid and Stable, i.e. transfer – read or write - can take place

Most 8 Bit Microprocessors use Synchronous data transfers for simplicity, that is:-

The CPU issues a valid address and the memory system has to respond to the rea/write request within a specified time period (measured in clock cycles) otherwise the transfer will **fail**. That is, there is no *handshake* back from the memory to the CPU to indicate if/when the memory has performed the requested operation. The system designer has to calculate the 'read' and 'write' access and cycle times of the memory to make sure it is compatible with the CPU. This is a detailed subject for a later lecture.

Typical 16k Byte Memory Block Interfaced to the 6811