CHAPTER X

MEMORY SYSTEMS

READ MEMORY NOTES ON COURSE WEBPAGE (BELOW)

READ pp 399-426 (285-310), MANO AND KIME & www.ece.gatech.edu/academic/courses/ece2030/readings/memory/memory.pdf

MEMORY SYSTEMS

INTRODUCTION

- A number of different types of memories and programmable logic devices exist.
  - Random-access memory (RAM)
  - Read-only memory (ROM)
  - Programmable logic devices (PLDs)
    - Programmable logic arrays (PLAs)
    - Programmable array logic (PAL)
    - Programmable gate arrays (PGAs)
  - Programmable sequential arrays (PSAs)
  - Field-programmable gate arrays (FPGAs)
- Due to time limitations, we will only cover RAM.

Memories (RAM, ROM)

Two-level combinational networks (PLDs: PLAs, PAL, PGAs)

Two-level (PSAs) / Multi-level (FPGAs) — combinational and sequential networks

MEMORY SYSTEMS

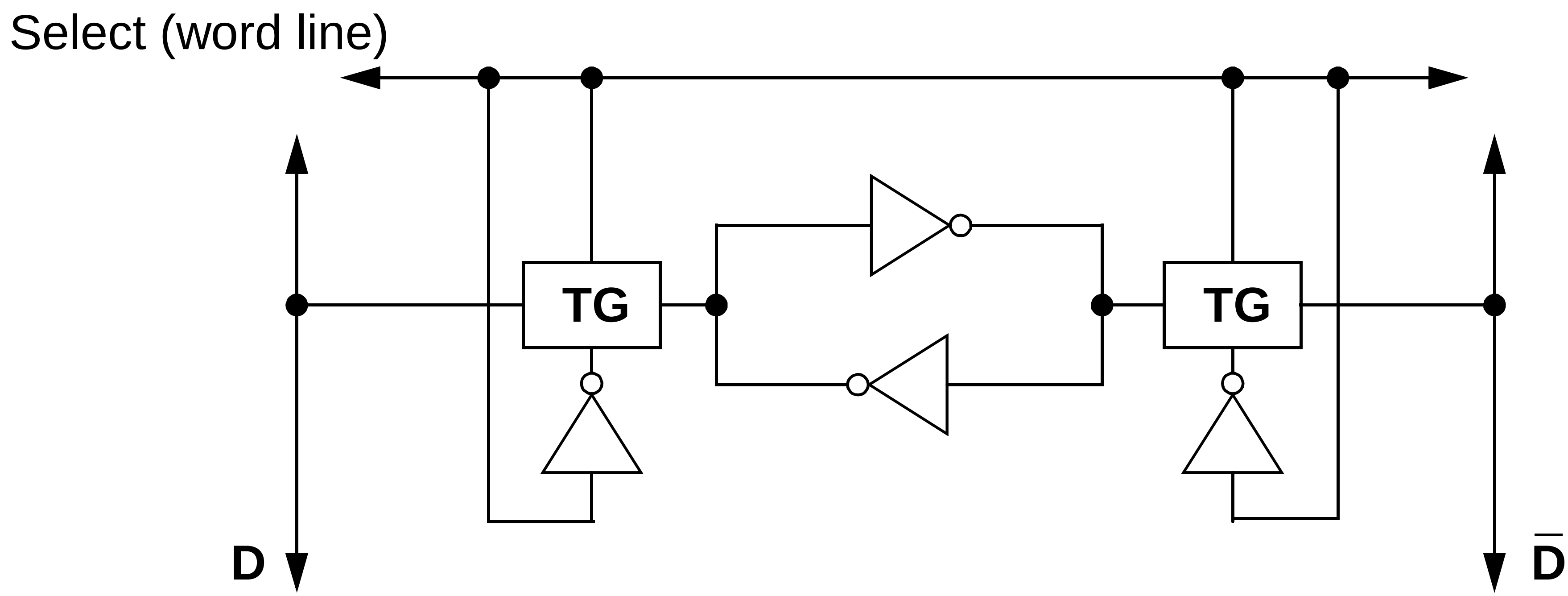
TYPES OF RAM

- Two main categories of random-access memory (RAM) exist.
  - Static memory or static RAM (SRAM)
    - Information bits are latched such as with a latch or a flip-flop.
    - Typical SRAM implementations require 4 to 6 transistors.
  - Dynamic memory or dynamic RAM (DRAM)
    - Information bits are stored in the form of electric charges on capacitors.
    - The capacitors will discharge over time.
    - Refreshing the memory cell is required before the capacitor has discharged to much of the electric charge.
    - Most DRAM implementations use 1 transistor and 1 capacitor.

STATIC RAM

SRAM CELLS (1)

- An inefficient SRAM bit cell can be formed as follows.



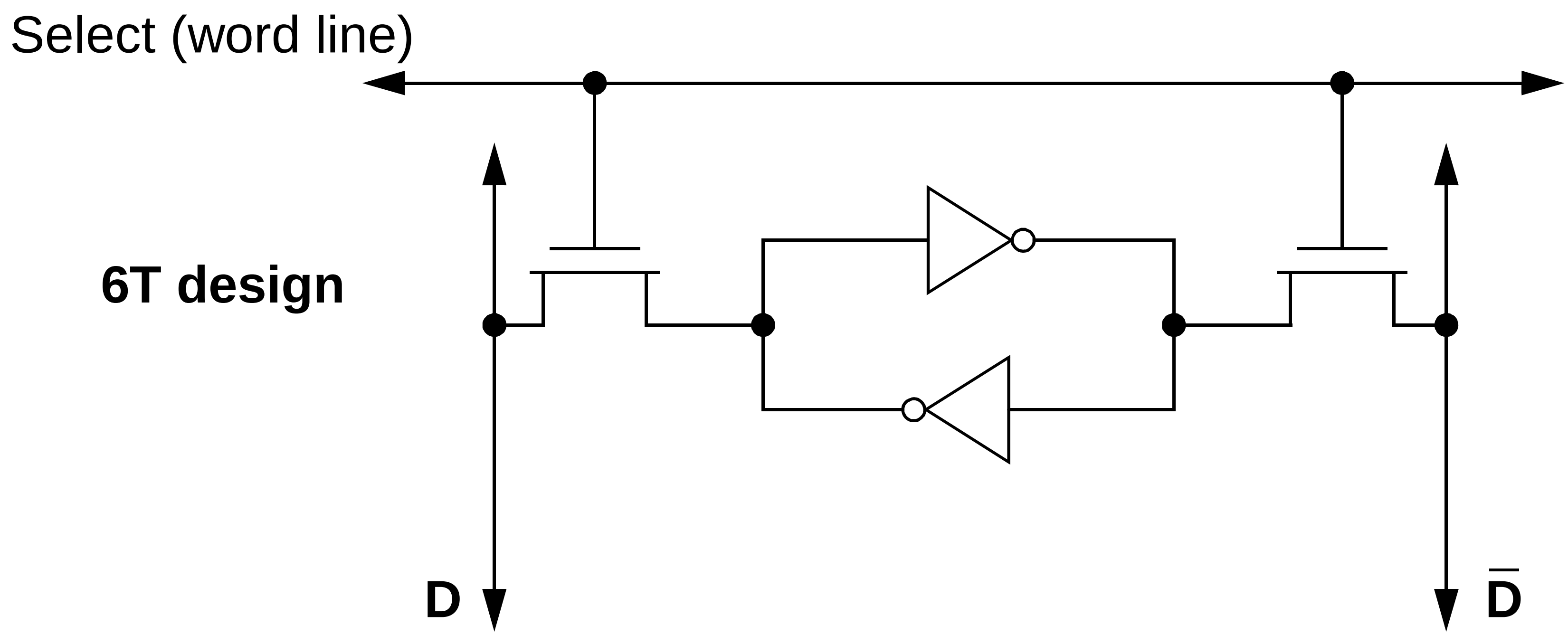
- How many transistors required for this design?
  - 2*4 for inverters + 2*2 for TGs = 12 transistors.
  - Very expensive in terms of silicon real estate!!!

STATIC RAM

SRAM CELLS (2)

- The structure for a 6 transistor implementation of an SRAM 1-bit cell is as follows. (We will refer to this as the "6T" design)

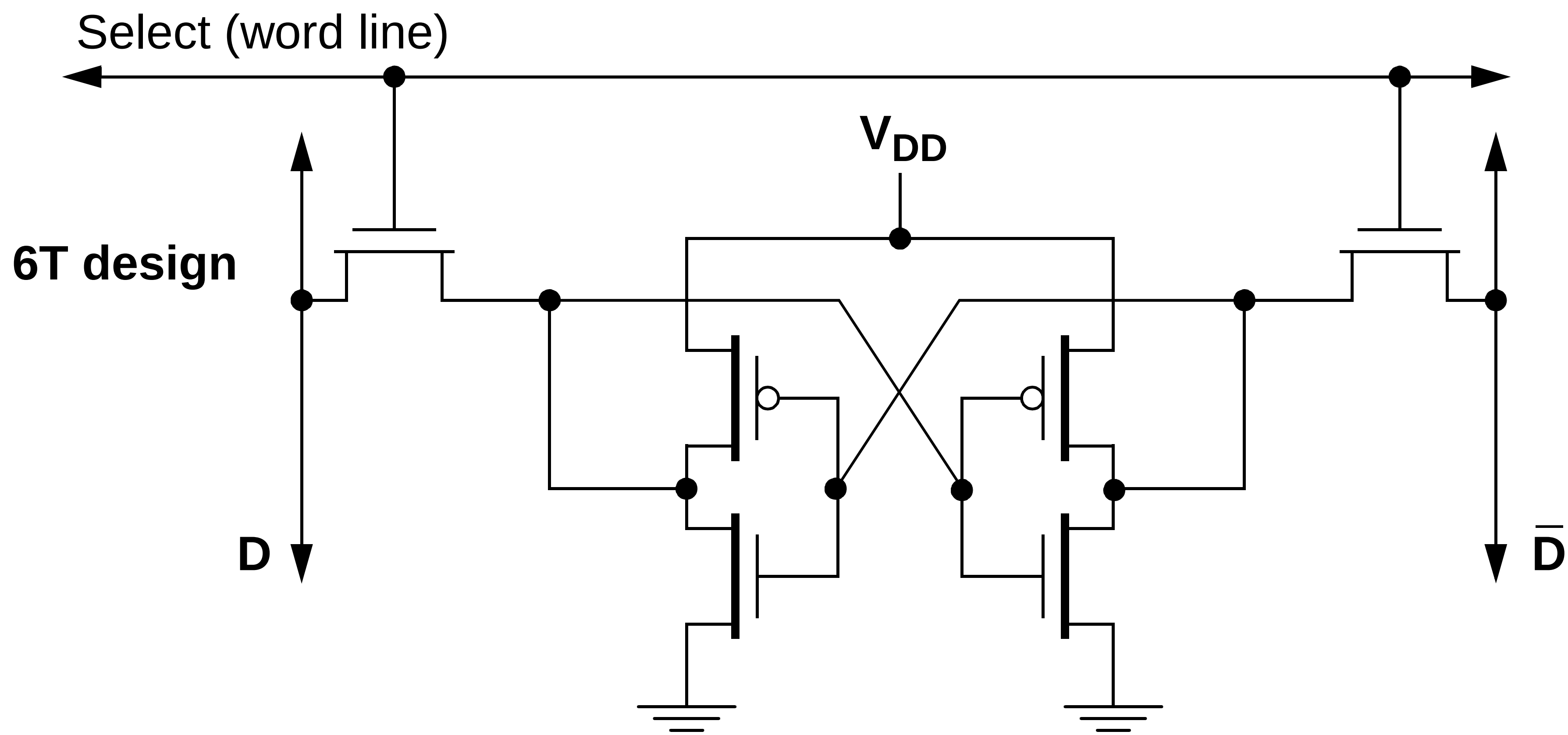
6T design



- The select, or word line, chooses the bit cell when high.
  - When selected, the new $\mathbf{D}/\overline{\mathbf{D}}$ is latched into the feedback loop.

STATIC RAM
SRAM CELLS (3)

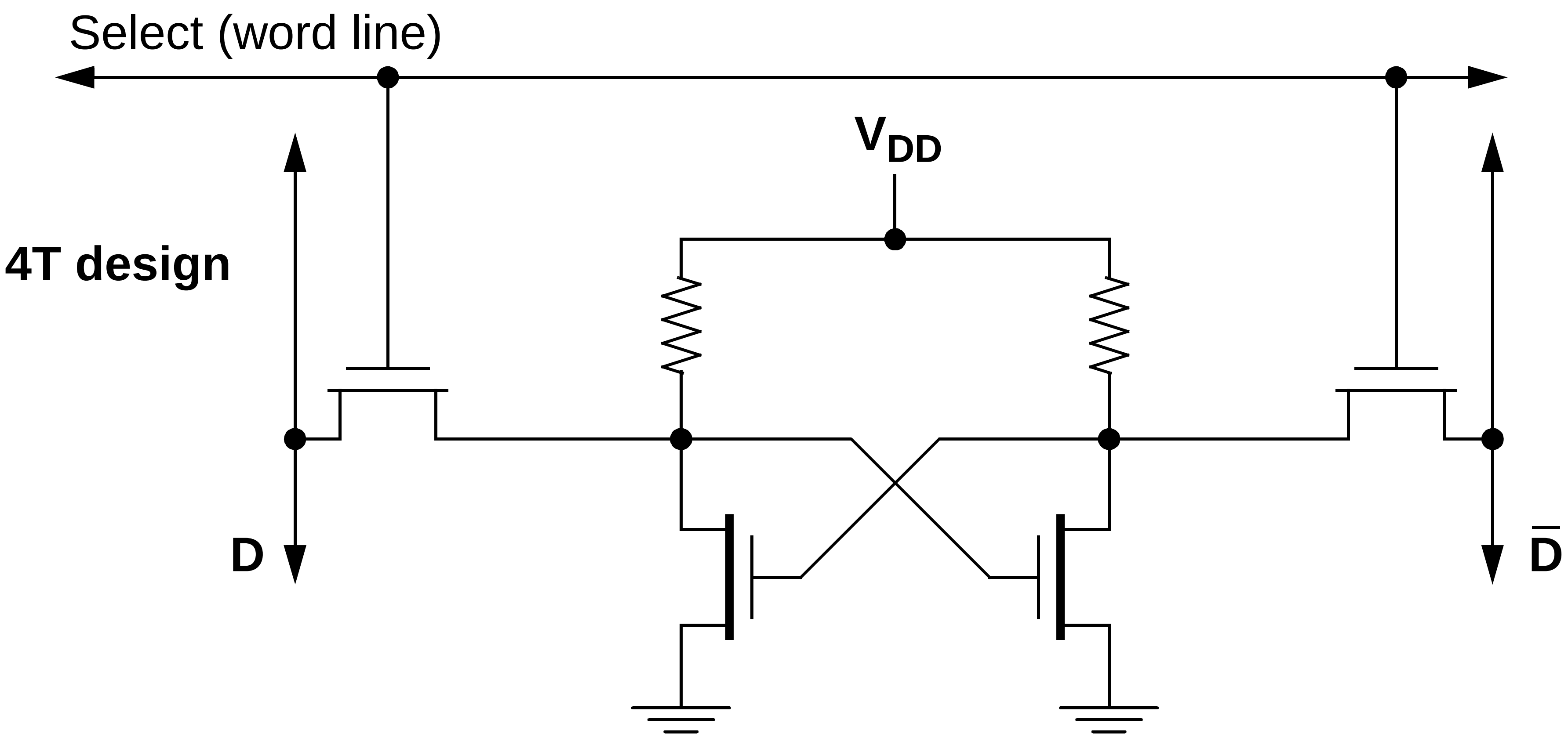
- Of course, the previous SRAM cell structure can be drawn as follows, replacing each inverter with 2 transistors.



STATIC RAM

SRAM CELLS (4)

- A 4 transistor design for an SRAM bit cell is as follows.

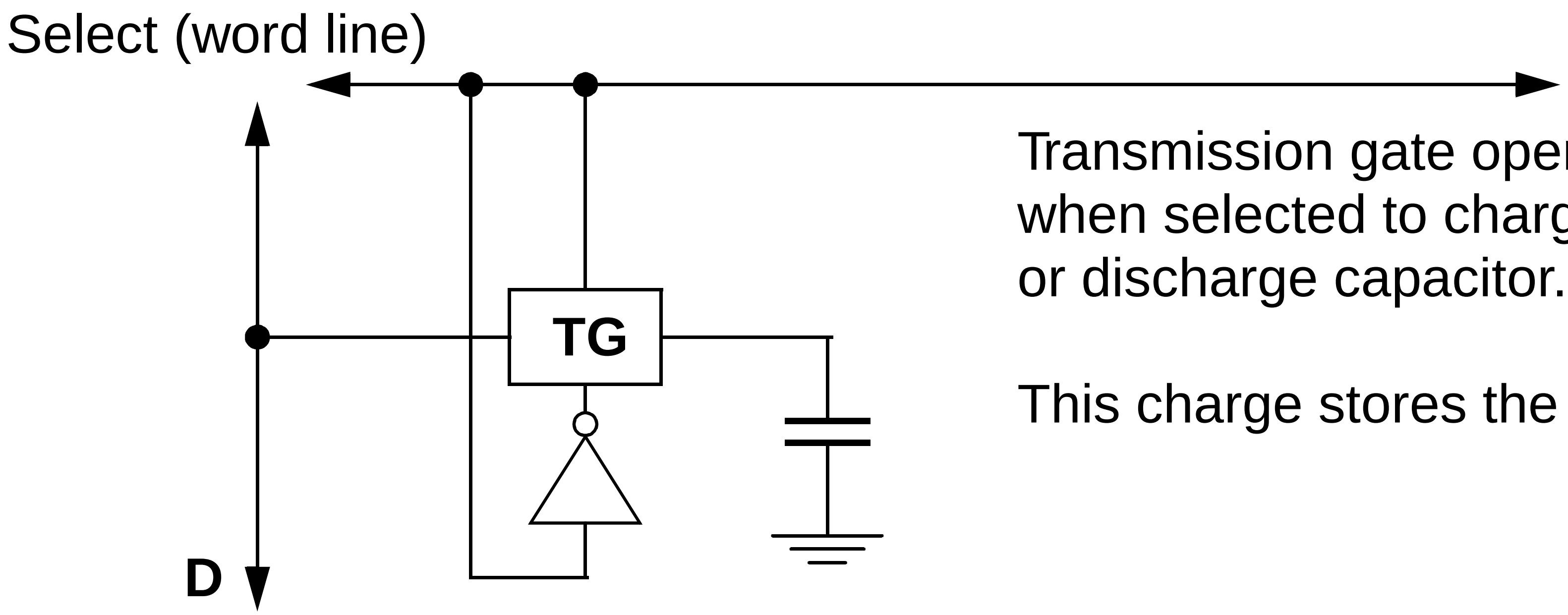


- Notice replacement of pMOS transistors with load resistors.
- This is for your own information. We won't be testing on the 4T design.

DYNAMIC RAM

DRAM CELLS (1)

- A dynamic RAM cell stores the bit as a charge in a capacitor.
- This bit must be refreshed periodically (>100s of times a second).



Transmission gate opens when selected to charge or discharge capacitor.

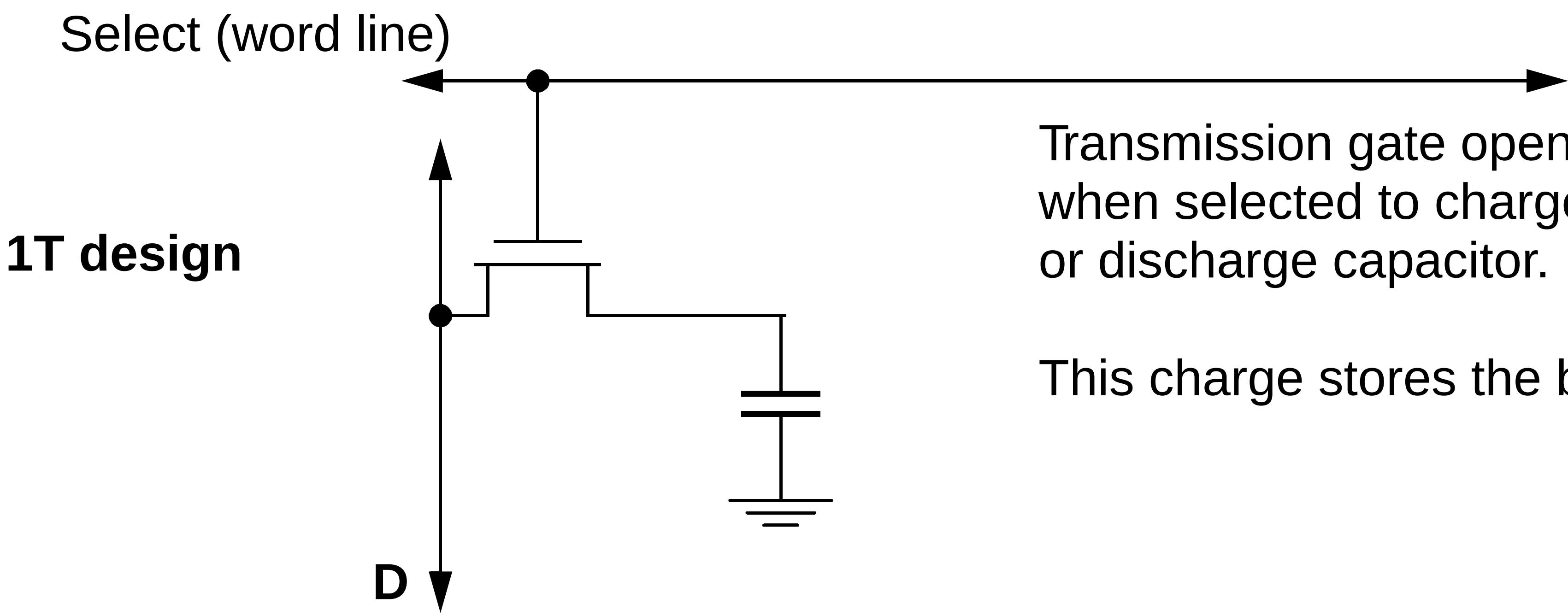
This charge stores the bit.

- How many transistors required for this design?
  - 2*1 for TG and 2*1 for inverter = 4 transistors.
  - Still expensive considering the extra refresh circuitry required!

DYNAMIC RAM

DRAM CELLS (2)

- The capacitor charging structure can be simplified as follows.



Transmission gate opens when selected to charge or discharge capacitor.

This charge stores the bit.

- This structure for a DRAM bit cell is what is used in practice in real implementations.
  - Very little chip real estate is used!!!

MEMORY UNITS

SPECIFICATION

- Having developed bit cells, either SRAM or DRAM bit cells, they can now be pieced together forming a memory unit.

- What do we want to specify in the design of a memory unit?
  - The number of bits.
    - This gives the total number of bits that the memory unit can store.
  - The grouping of bits into words.
    - Accessing 1 bit at a time might be inconvenient, so, grouping bits into words is often done.
    - Common examples of word bit sizes are 4, 8, 16, 32, and 64.
  - The number of words in the memory unit (addressable words).
    - This is a function of the word size and total number of bits.

MEMORY UNITS

DESCRIPTION

- In describing the capacity of a memory unit, the following is used
  - # addresses x word size
    - Example: **1Mx8**

- If a memory unit is described as 1Mx8, then it has
  - $\mathbf{1M} = \mathbf{2^{20}} = \mathbf{1048576}$ addresses,
  - **8** bits per word at each address location,
  - **8** data lines for the **8** bit words,
  - **20** address lines to specify the $\mathbf{1M} = \mathbf{2^{20}} = \mathbf{1048576}$ addresses,
  - and $(\mathbf{1048576})(\mathbf{8}) = \mathbf{8388608}$ bits in the entire memory unit.

MEMORY UNITS

DESCRIPTION EXAMPLES

- Some further examples of memory descriptions are given below.
  - Note that the last four columns are all described with the information in the first column.
  - Try to fill in the empy cells for the last two rows.

Memory	Total bits	# of addresses	# address lines	# data lines
1Mx8	8388608	1048576	20	8
1Kx4	4096	1024	10	4
2Mx4	8388608	2097152	21	4
4Mx1	4194304	4194304	22	1
2Mx32	67108864	2097152	21	32
16Kx64				
8Mx8				

MEMORY UNITS

DESCRIPTION EXAMPLES

- Some further examples of memory descriptions are given below.
  - Note that the last four columns are all described with the information in the first column.
  - Try to fill in the empy cells for the last two rows.

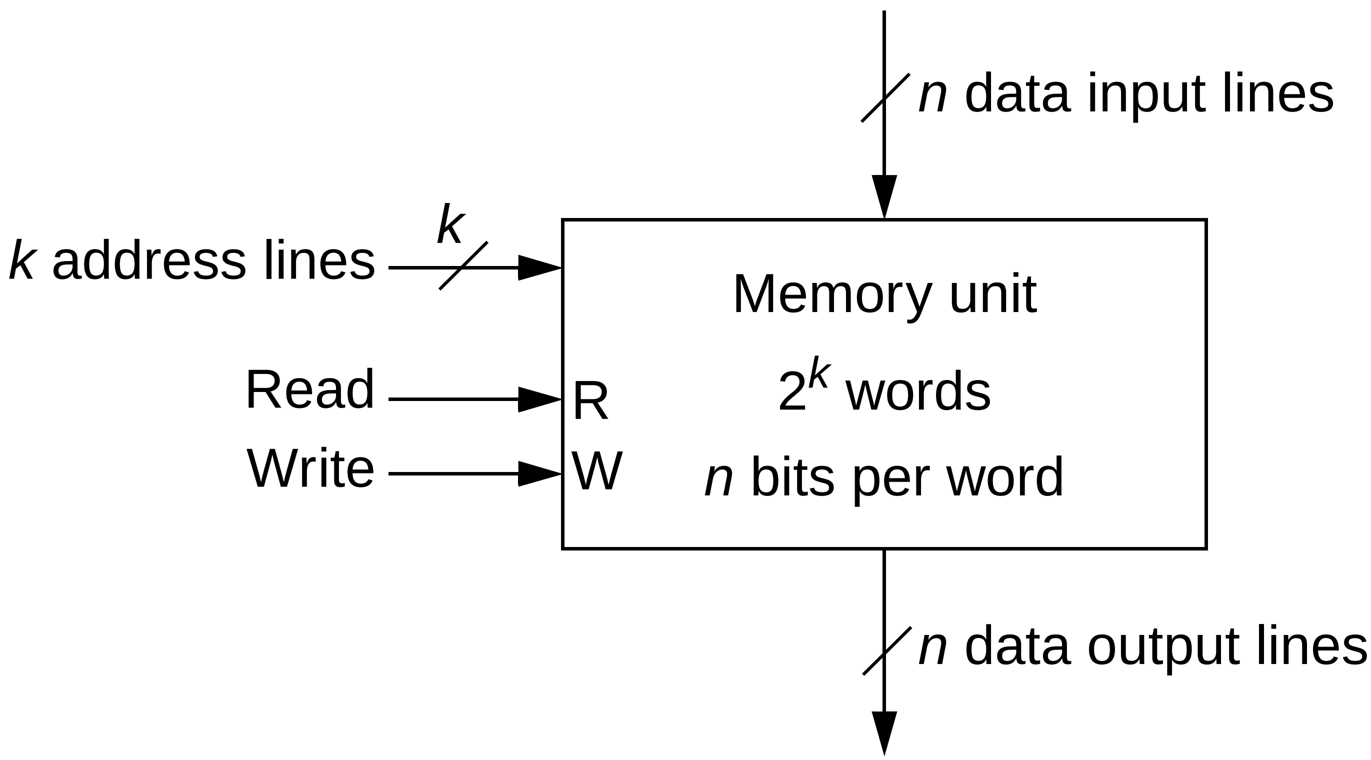
Memory	Total bits	# of addresses	# address lines	# data lines
1Mx8	8M	1M	20	8
1Kx4	4K	1K	10	4
2Mx4	8M	2M	21	4
4Mx1	4M	4M	22	1
2Mx32	64M	2M	21	32
16Kx64				
8Mx8				

Binary-Decimal Approximation: K = 1000 = 2^10 M = 1,000,000 = 2^20 G = 1,000,000,000 = 2^30

MEMORY UNITS

BLOCK DIAGRAM (1)

- Below is a general block diagram for a memory unit.

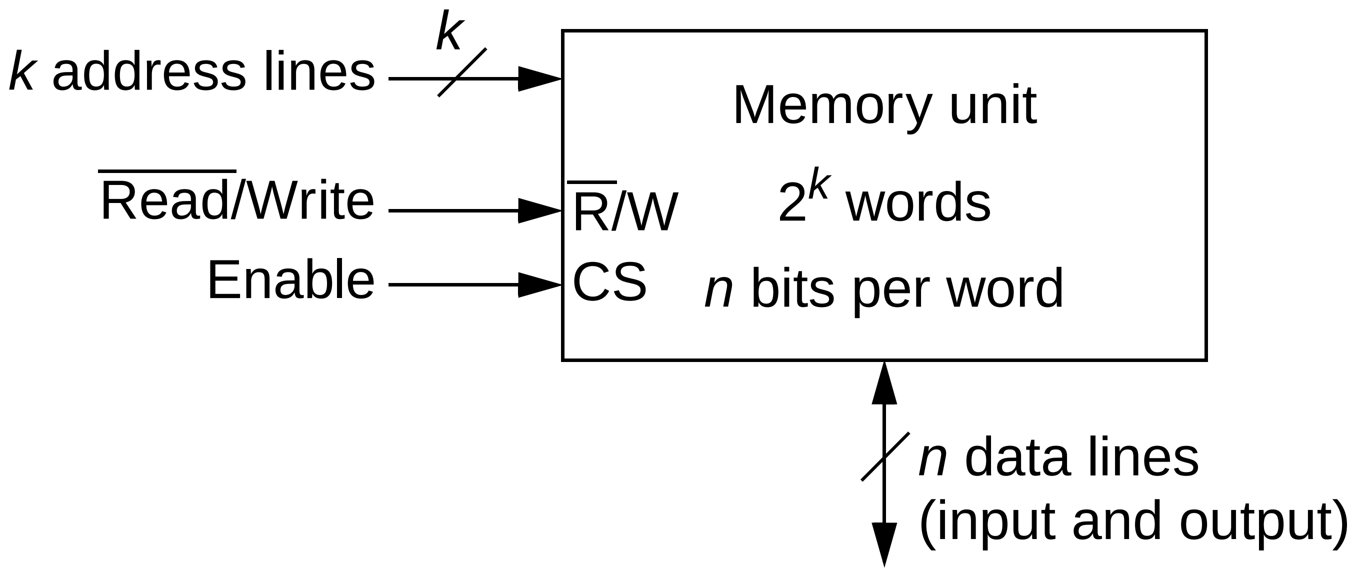


- The k address lines access a word in the memory for input or output.
- To simplify drawing, we now form buses of *n* (or *k*) lines.

MEMORY UNITS

BLOCK DIAGRAM (2)

- To conserve pins, the following layout is more common in practice.

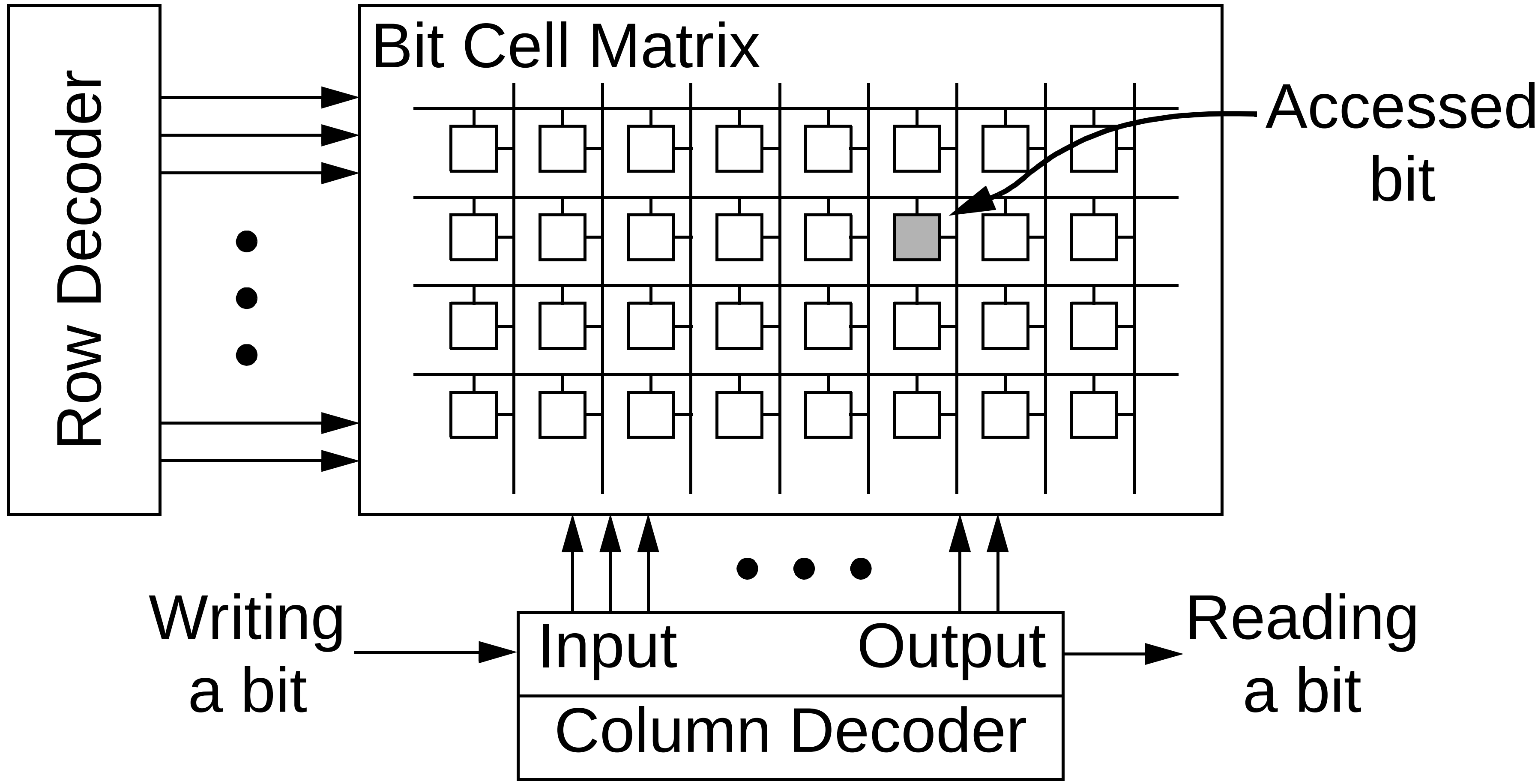


- The data lines are both input and output lines (not simultaneously).
  - This is done by using tristate buffers to form a tristate bus (or sometimes referred to as a three-state bus).

MEMORY UNITS

INTERNAL STRUCTURE

- The bit cells are arranged in matrix. (more efficient!)
- Row and column decoders access specific bit cells.



MEMORY UNITS

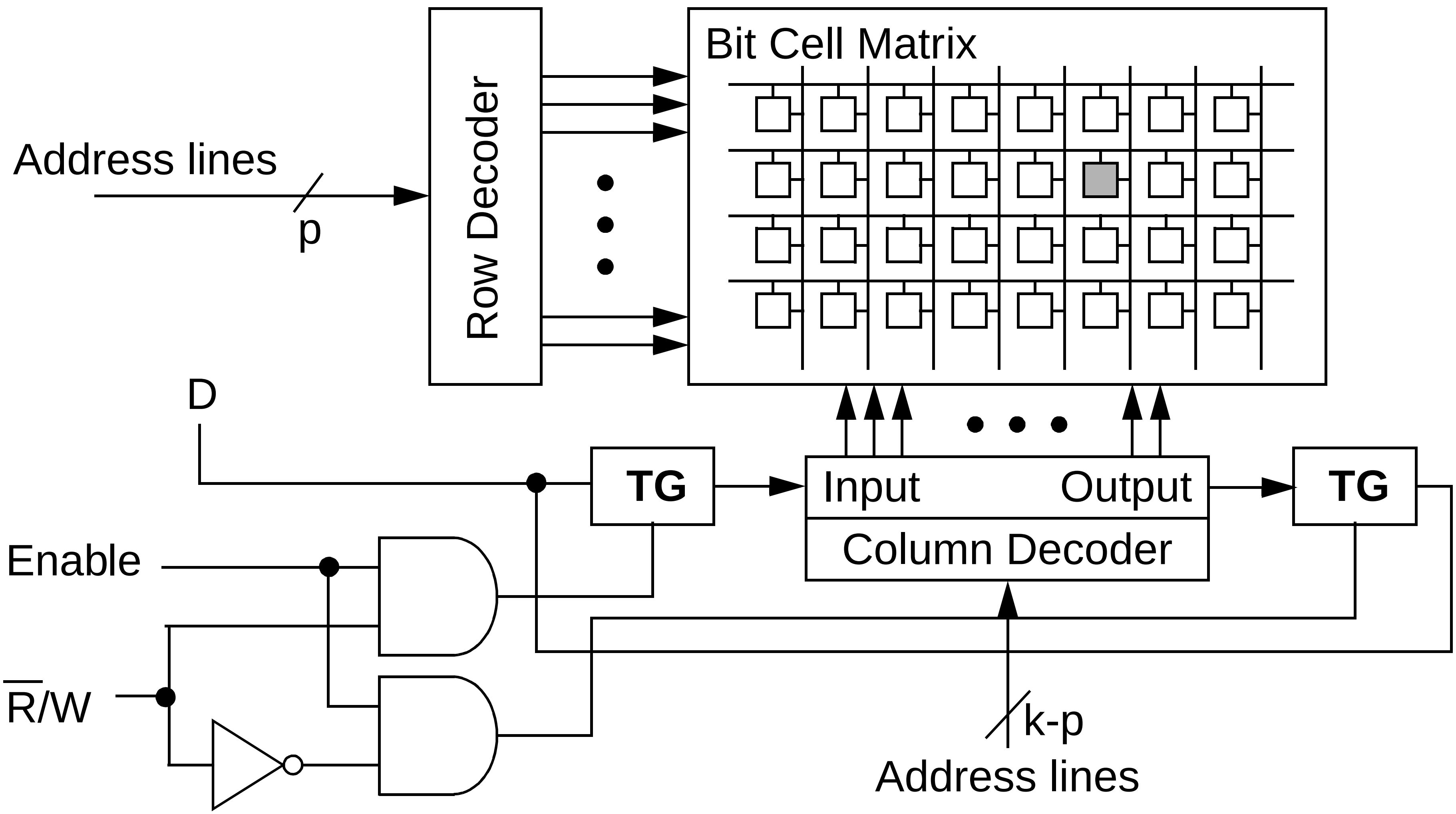
READ/WRITE OF 1-BIT

- In read mode:
  - Row decoder "activates" all bit cells in that row.
    - Each bit cell in the row outputs their stored bit.
  - Column decoder takes the bit from only one column of the activated row.
- In write mode:
  - Row decoder "activates" all bit cells in that row.
    - Each bit cell in the row effectively outputs their stored bit.
  - Column decoder selects the appropriate column and writes the input bit.
    - SRAM: This writing is done by "overpowering" what is being read by the bit cell with a stronger voltage/current.
    - DRAM: This writing is done by recharging the capacitor for writing a **1** or discharging the capacitor for writing a **0**.

MEMORY UNITS

CONTROL LINES (1)

- Now include *k* address lines, 1-bit data line, Enable, and ~Read/Write.



MEMORY UNITS

CONTROL LINES (2)

- Things to note about the previous implementation.
  - The ***k* address lines** are split into two parts (not necessarily equal parts)
    - ***p*** of the *k* address lines are sent to the **row decoder**.
    - ***k-p*** of the *k* address lines are sent to the column decoder.
  - There is only **one bit line** for **input/output**, D.
    - If we desire multiple bits, say *n* bits, for each address location, the entire structure must be duplicated *n* times.
  - The **enable line**, or often called chip select (CS), either allows the transmission gates to be closed or prevents them from being closed. This makes it so that the D can be in one of three modes;
    - reading, writing, or high impedence.
  - The **~read/write** line controls which transmission gate is closed.

MEMORY UNITS

BITS FOR ROWS/COLUMNS

- In general, given a certain size memory chip, such as a 1Mx1 memory chip, we would not know how the internal matrix is configured.
  - For a **1Mx1 memory chip**, we know it has **20 address lines** (for our purposes in any case, there are exceptions in the real world). Any combination of address lines for the row and column decoder could be used to form the matrix.
    - Example: **10 row** address lines and **10 column** address lines for a **1024x1024 matrix** of bit cells.
    - Example: **12 row** address lines and **8 column** address lines for a **4096x256 matrix** of bit cells.
    - Example: **5 row** address lines and **15 column** address lines for a **32x32768** matrix of bit cells.

MEMORY UNITS

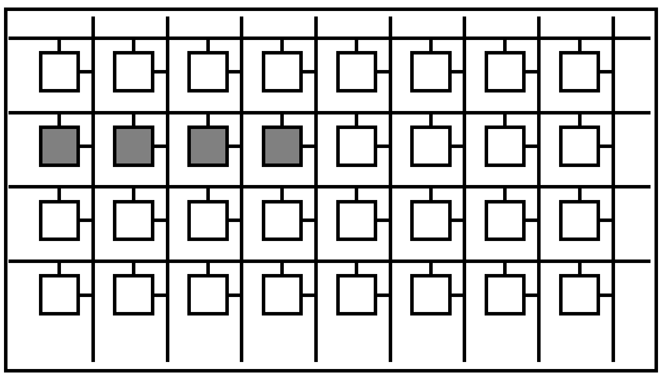
MULTIPLE BITS (WORDS)

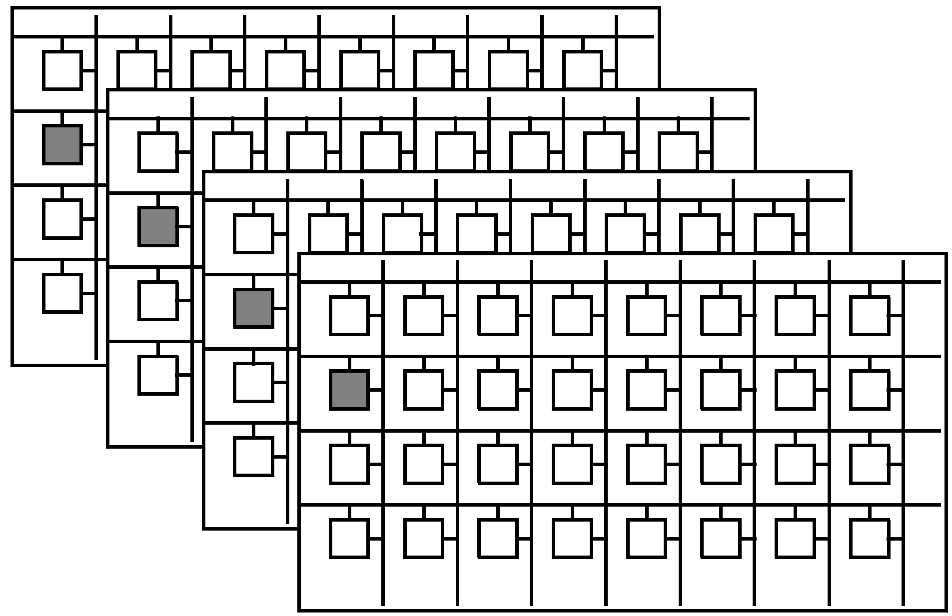
- As noted, the described internal bit cell matrix structure accesses only 1 bit at a time.
- Multiple bits to form an *n*-bit word could be accessed in a few different methods.
  - One method is to have the **column decoder** select a set of ***n* columns simultaneously** to form the word.
    - This only works if the entire word is stored in one row. Hence, there should be a multiple of *n* columns in the bit cell matrix.
    - For instance, the given bit cell matrix with 8 columns could easily have words of size 1, 2, 4, or 8.
  - Another method, though arguably very similar, is to **duplicate** the entire **bit cell matrix *n* times** to form the *n*-bit word.

MEMORY UNITS

MULTIPLE BITS (WORDS)

- The following illustrates these approaches for accessing a 4-bit word.





- On the left, **four column bits** from the **second row** are selected to form the **4-bit word**.
- On the right, the **first column bit** from the **second row** for **4 duplicates** of the bit cell matrix are accessed to form the **4-bit word**.

BUILDING SYSTEMS

DESIGNING MEMORY SYSTEMS

- What if we don’t have the right type of memory chips to build a desired computer system?
- Must learn to use combinations of existing memory chips to form a memory system according to specifications.
  - Example: We want a **1Mx8** memory system but can only cheaply buy 1Mx4 memory chips. What to do?
    - Become Bill Gates so you can afford it.
    - Get a better job.
    - Go back to storing your valuable information on little pieces of paper in your pocket. (be careful of washing machines!!!)
    - Design a memory system that uses **1Mx4** memory chips but logically forms a **1Mx8** memory system.

BUILDING SYSTEMS

EXAMPLE #1

- Build a 1Mx8 memory system using 1Mx4 memory chips.
  - First identify the specifications for the **1Mx4** memory chips and the desired **1Mx8** memory system?

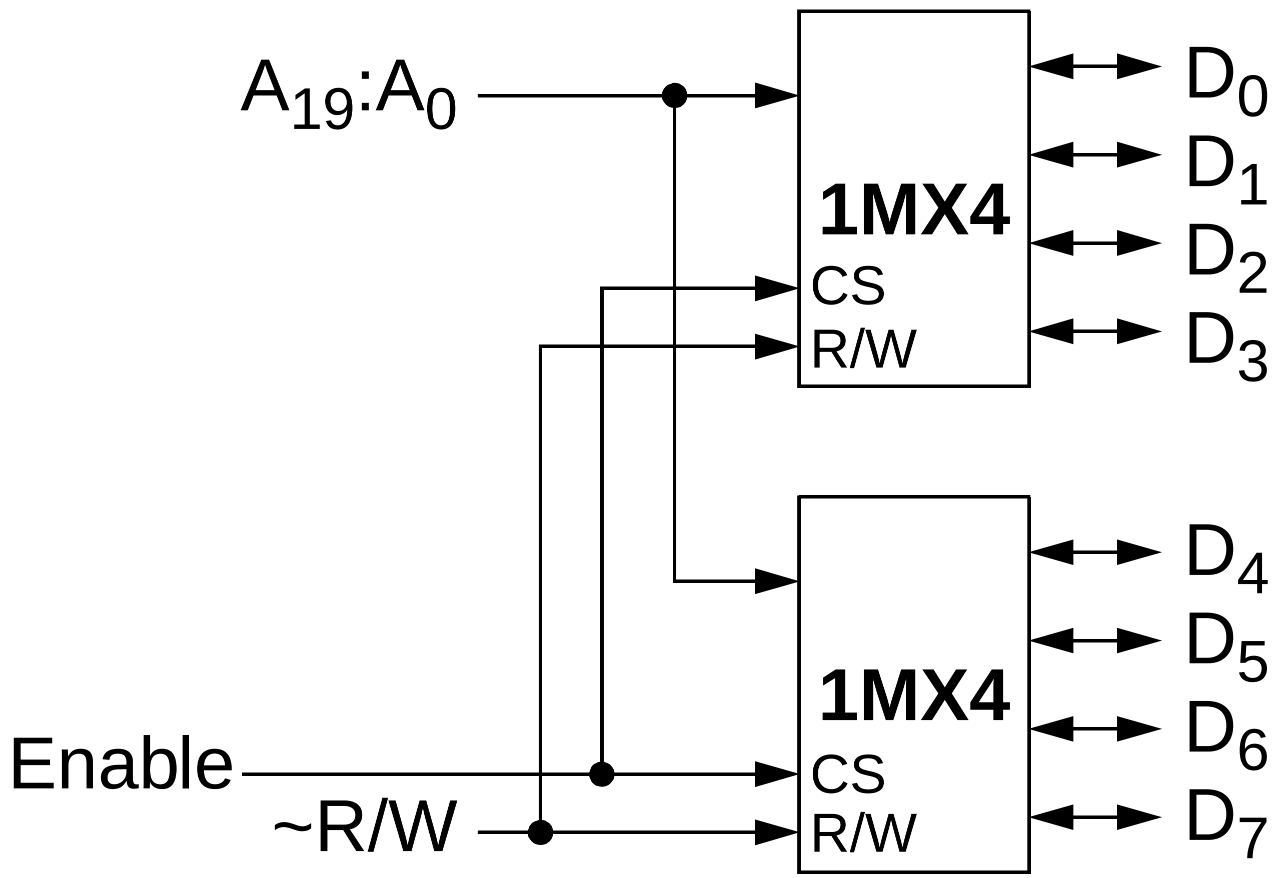
Memory	Total bits	# of addresses	# address lines	# data lines
1Mx8	8388608	1048576	20	8
1Mx4	4194304	1048576	20	4

  - As the table shows, the # of addresses and # of address lines are the same for both. So, we do not have to change that.
  - The number of data lines for the **1Mx8** are double that of the **1Mx4** as well as the total # of bits stored in the memory.
    - Therefore, we require **two 1Mx4** memory chips arranged with the **same address lines** and **concatenated data lines**.

BUILDING SYSTEMS

EXAMPLE #1

- Forming the described 1Mx8 memory system with 1Mx4 chips, the following 1Mx8 memory system can be designed.



BUILDING SYSTEMS

EXAMPLE #2

- Build a 2Mx4 memory system using 1Mx4 memory chips.
  - First identify the specifications for the **1Mx4** memory chips and the desired **2Mx4** memory system?

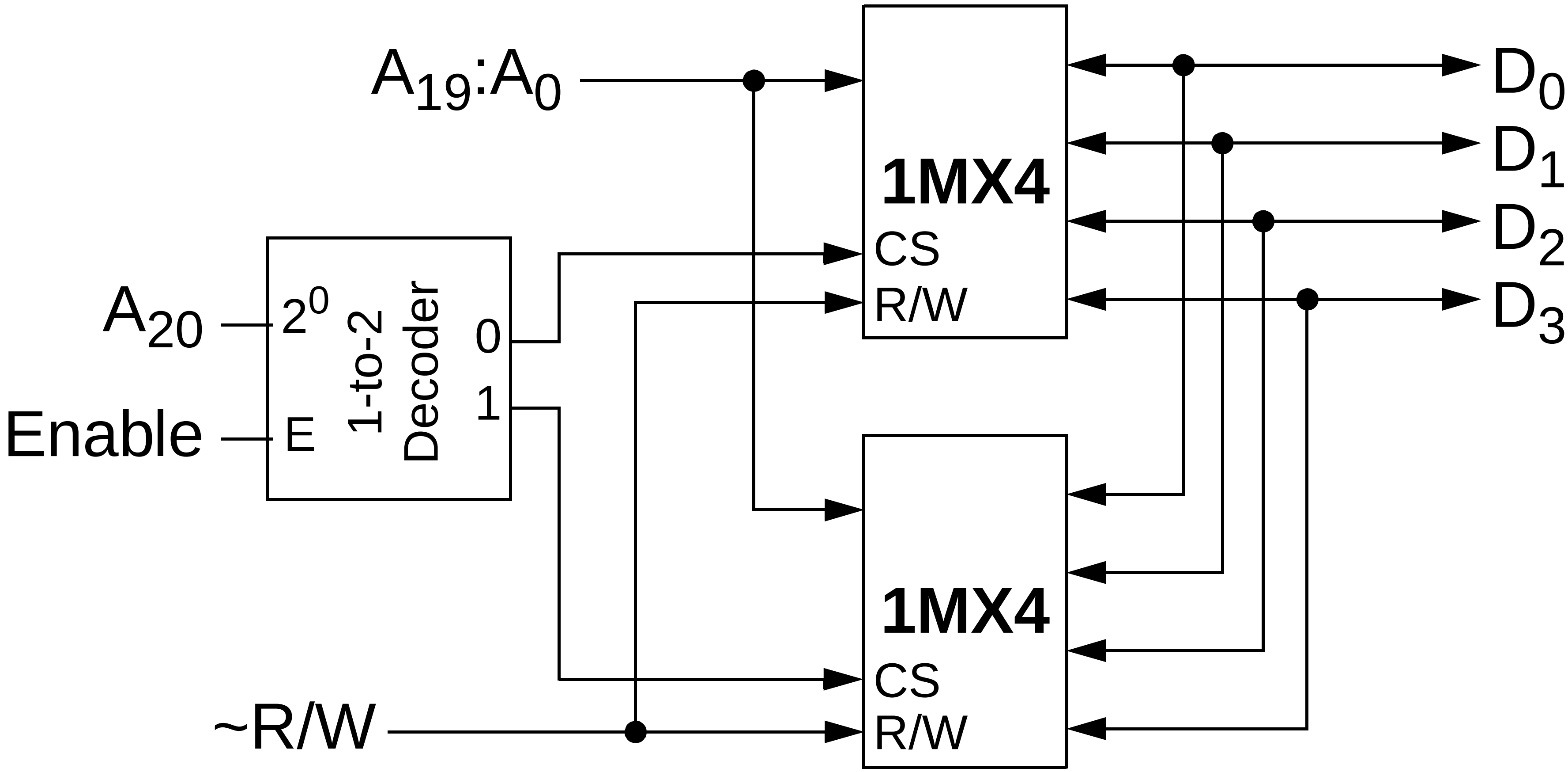
Memory	Total bits	# of addresses	# address lines	# data lines
2Mx4	8388608	2097152	21	4
1Mx4	4194304	1048576	20	4

  - As the table shows, the # of data lines are the same for both.
  - There is an extra address line in the **2Mx4** giving double the # of addresses and double the # of bits
    - Therefore, we require **two 1Mx4** memory chips arranged where **one address line** is used to **decode** which **1Mx4** chip is **enabled**.

BUILDING SYSTEMS

EXAMPLE #2

- The following implements the desired 2Mx4 using 1Mx4 memory.



- Notice that A_{20} is used to activate one of the **1Mx4** memory chips.

MEMORY MODEL

PROGRAMMER'S MODEL

- Goal is to abstract the memory model so that a programmer of the system does not need to know the physical layout of the memory.
  - Below is an example 2^{32} addressable memory map of byte. This, for instance, could be implemented with a **4Gx8** memory system.

0x00000000	0x43
0x00000001	0x78
0x00000002	0x12
0x00000003	0x04
	⋮
0xFFFFFFFF	0x7c

MEMORY MODEL

ENDIAN BYTE ORDERING (1)

- How should a multiple byte word be stored?
- The two most common orderings are
  - Big endian
    - The address is of the most significant byte location.
    - Sun Workstations and Macs use this ordering.
  - Little endian
    - The address is of the least significant byte location.
    - Intel x86 architectures use this ordering.
- Origins of the terms big and little endian.
  - Gulliver's Travels: Feud between the two mythical islands, Lilliput and Blefescu, over the correct end (big or little) at which to crack an egg.

MEMORY MODEL

ENDIAN BYTE ORDERING (2)

- An example of a 4-byte word stored in big endian order
  - 0x12345678 would be stored as

Address	Contents
0x10010000	0x12
0x10010001	0x34
0x10010002	0x56
0x10010003	0x78

12345678

- An example of a 4-byte word stored in little endian order
  - 0x12345678 would be stored as

Address	Contents
0x10010000	0x78
0x10010001	0x56
0x10010002	0x34
0x10010003	0x12

78563412

MEMORY MODEL

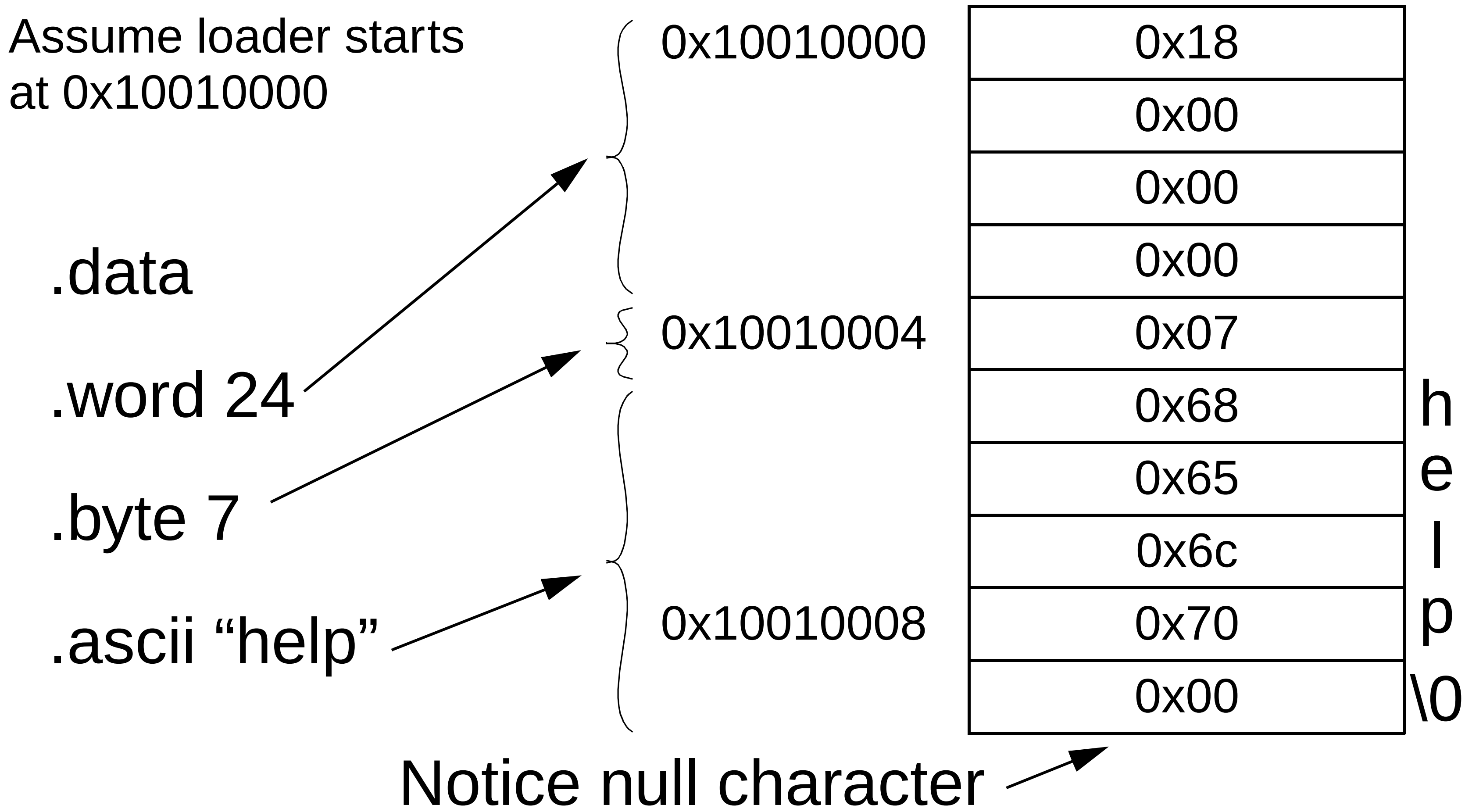
ASSEMBLER DIRECTIVES

- Let's assume we have a memory system with 32 bit words using little endian byte ordering.
  - .word n_{10}
    - Store the value n_{10} in memory as a 32 bit word binary value.
  - .byte n_{10}
    - Store the value n_{10} in memory as a byte value (8 bits).
  - .asciiz "string"
    - Store the ASCII string in memory with a null character.
  - .space n_{10},
    - Skip the next n bytes.
  - .align k
    - Force loader to go to next 2^k byte boundary.

MEMORY MODEL

EXAMPLE DIRECTIVE USE (1)

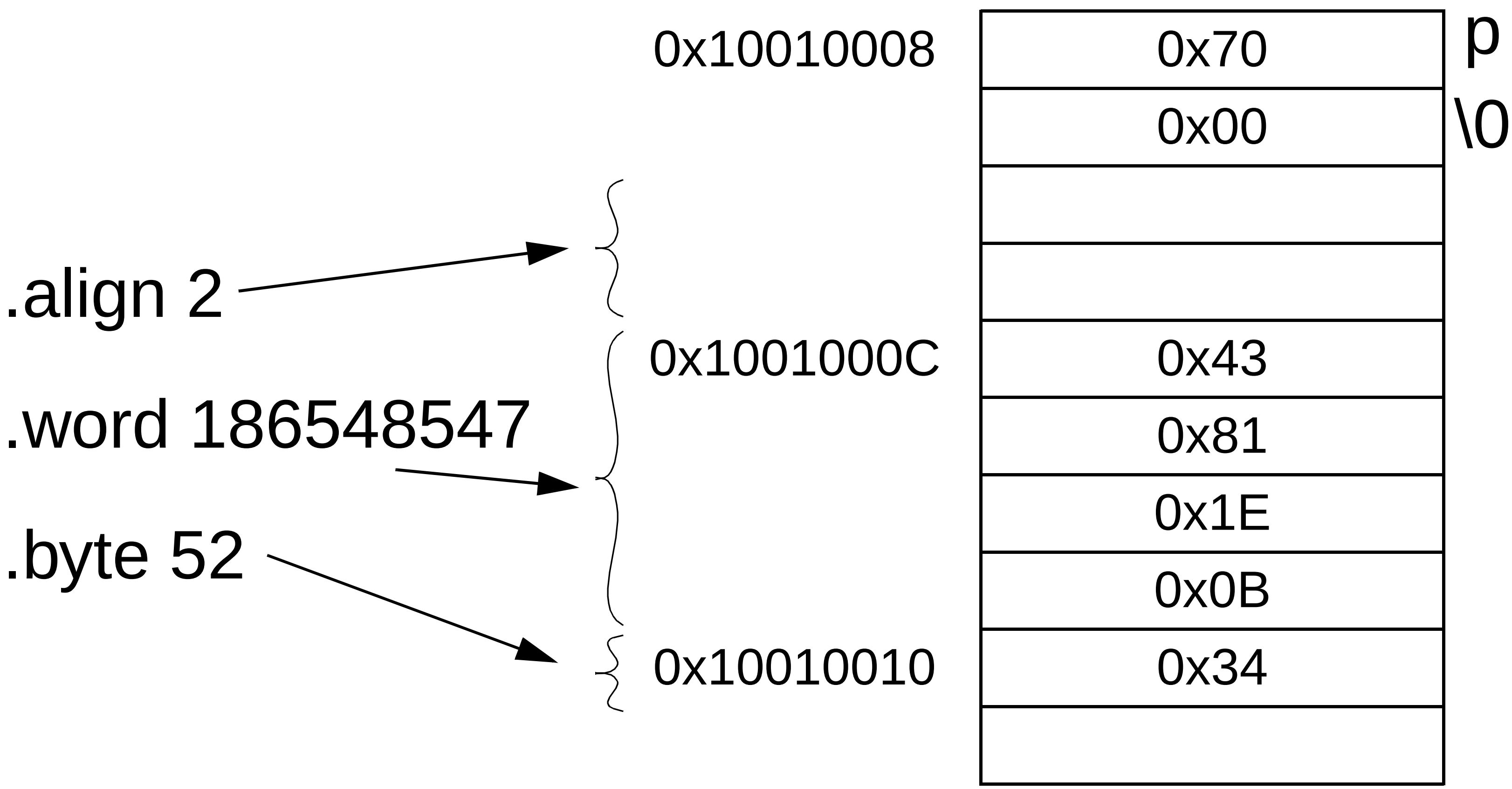
- One example assembly memory directive set could be as follows.



MEMORY MODEL

EXAMPLE DIRECTIVE USE (2)

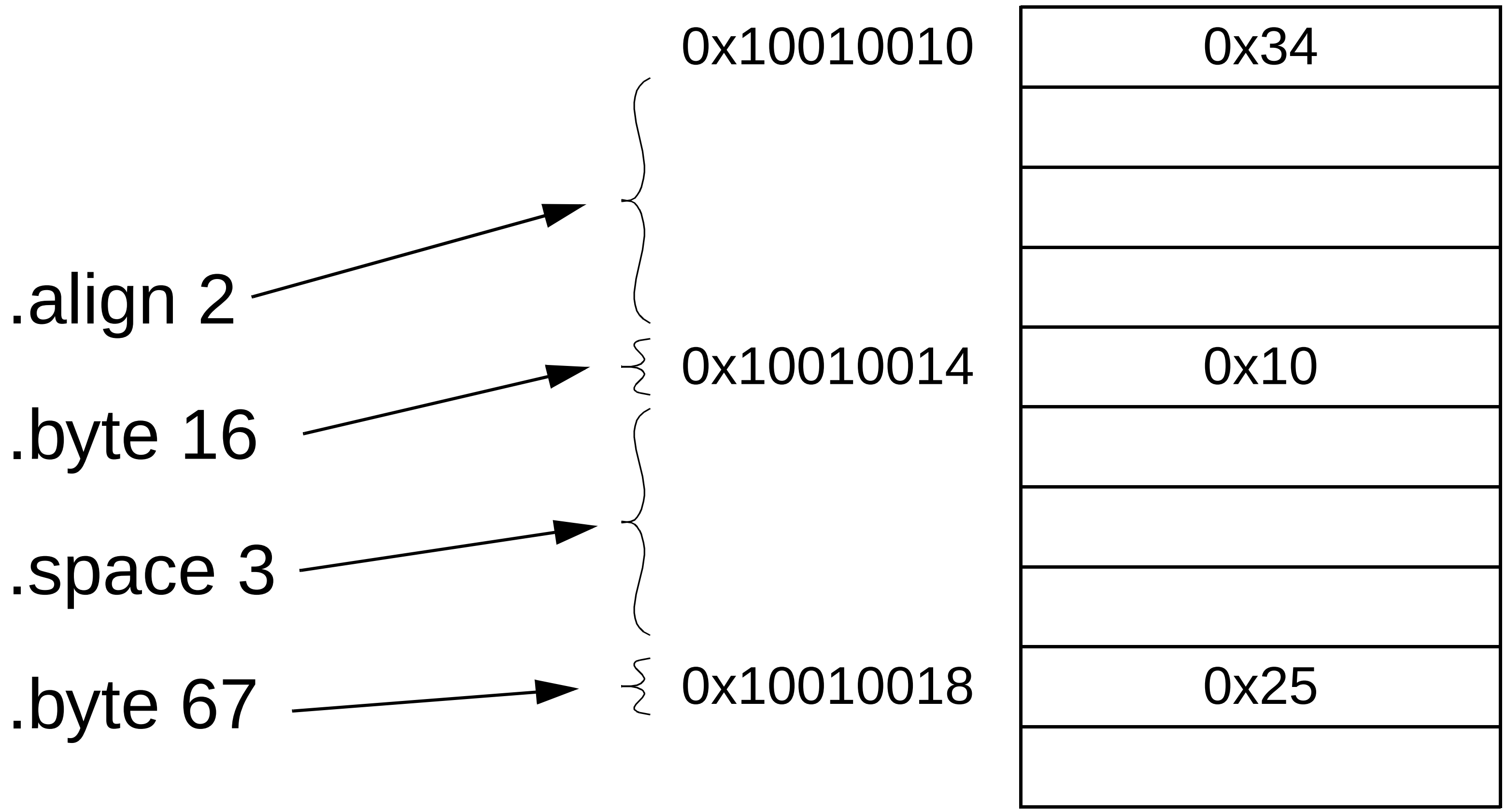
- Continuing with the previous example.



MEMORY MODEL

EXAMPLE DIRECTIVE USE (3)

- Continuing with the previous example.



- Notice the difference between align and space.