Memory Decode Logic Design

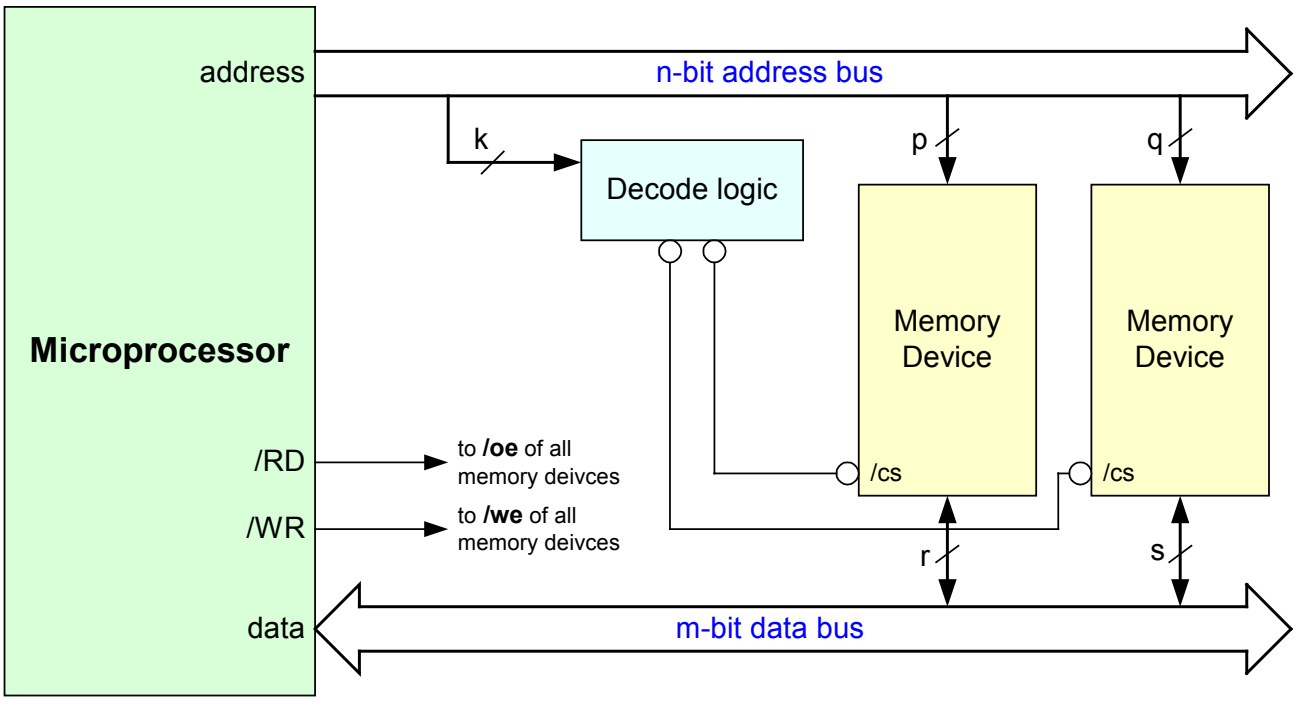
Dr. D. Capson
Electrical and Computer Engineering
McMaster University

Introduction

A microprocessor with an n-bit address bus can generate 2^n unique addresses with values 0 to $2^n - 1$. The purpose of decode logic is to interface memory devices with a microprocessor as shown in the diagram below. The input to the decode logic is k bits taken from the n-bit address bus. The address bus is unidirectional since the microprocessor *provides* addresses (it never receives them) as part of its operation of fetching and executing instructions. The output of the decode logic is a single bit "chip select" signal for each memory device. The decode logic may be viewed as 'monitoring' the address bus. When it 'sees' an address within the range it is designed to detect, it generates a corresponding output that is used to enable the appropriate memory device via its chip select (/cs) input. On some memory devices this is also called the 'chip enable' (/ce) input. Chip select and chip enable behave identically and are often used interchangeably.

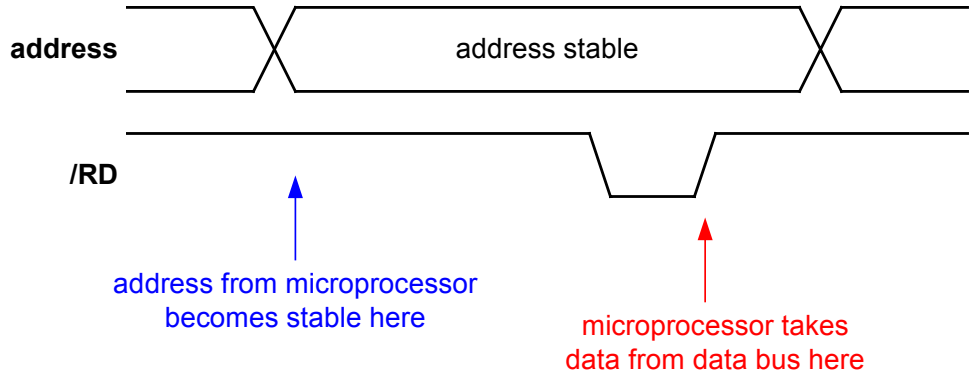
Note: The forward slash is front of a signal name is a way of indicating a LO-true signal. That is, the signal is normally HI when it is in its quiscient state and is considered asserted when LO.

The memory devices in a system may be of different types (mixtures of RAM and ROM) and may even be of different sizes and organizations. Note that in the diagram, the number of address bits **p**, **q** and **n** are not necessarily equal (although they can be). The data bus is bidirectional since data may be read from, or written to, a memory component depending on its type. The width of the data bus (**m**) need not be equal to the width of the data connections to/from the memory devices (**r** and **s** in the diagram).

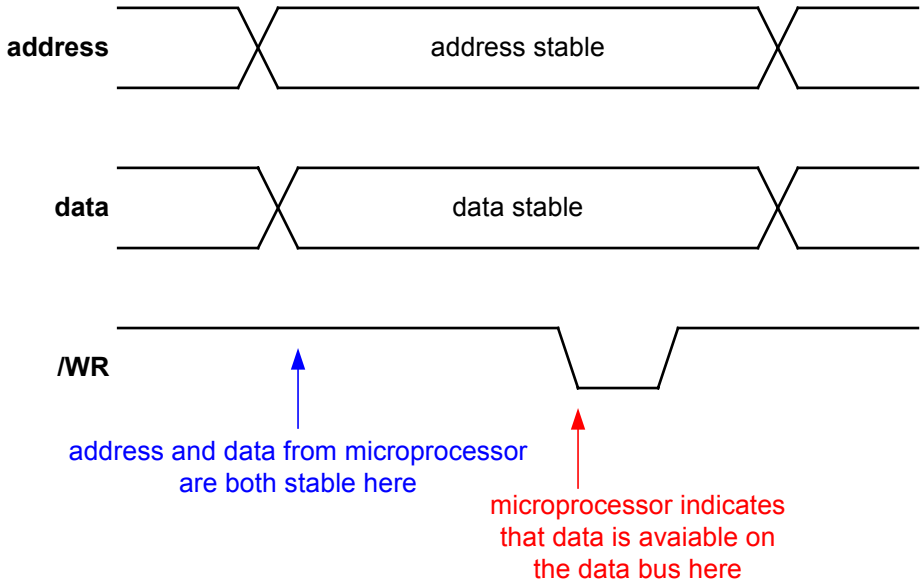


Microprocessors provide mutually exclusive signals /RD and /WR to coordinate access to memory (they are never asserted at the same time!) For reading from memory, the microprocessor places the desired address onto the address bus and then issues a read pulse /RD At the end of this read pulse, the microprocessor reads the contents of the requested memory location from the data bus where it is assumed to have been placed by the appropriate memory device. This is called a *memory read machine cycle*.

It is the role of the decode logic to ensure that the correct memory device is accessed and that it ***supplies*** data on the data bus for this cycle. The basic memory read machine cycle timing is as follows:



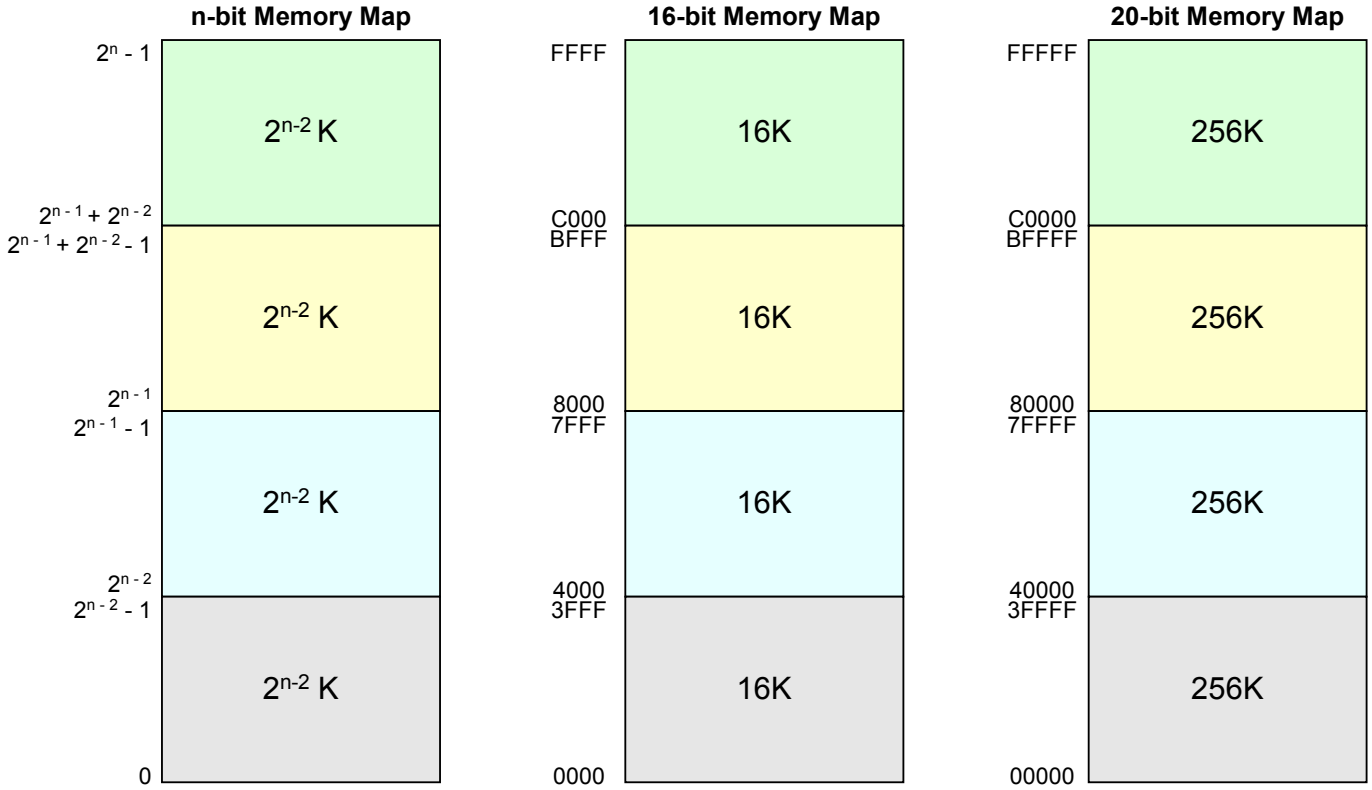
For a write operation, the microprocessor provides an address on the address bus, provides data on the data bus and then issues a write pulse /WR. This is called a *memory write machine cycle.* It is also the role of the decode logic to ensure that the correct memory device is accessed and that it ***takes*** data from the data bus at the end of this cycle. The basic memory write machine cycle timing is as follows:



All microprocessors use some form of this basic mechanism; it is the fundamental concept to understand the interfacing of memory and I/O devices to a microprocessor bus. The read and write machine cycles are a major factor in determining the overall performance of a system. There has been a wide variety of strategies developed by manufacturers of microprocessors and memory components to enhance this basic mechanism for improving performance.

Memory Maps

It is useful to graphically portray the entire range of possible addresses that can be generated by a microprocessor (also called the ***address space***) in the form of a ***memory map*** as in the following examples:



Consider a memory space with 16 address bits (i.e. n=16) as shown in the middle memory map above. Many small systems used in embedded applications use no more than 16 bits. This then defines an address space of 64K. Note that all of the addresses are shown in hexadecimal. We assume the individual bits of the address bus are labelled $\mathbf{A_{15}}$ (the msb) thru $\mathbf{A_0}$ (the lsb).

The minimum address is 0000H (16 binary 0's) and the maximum address is FFFFH (16 binary 1's) which is $2^{16} - 1$. The fact that the maximum address is shown at the top of the diagram is completely arbitrary and has no bearing on our discussion. The first address of the top half of the memory map is 8000H. *Every* address from this point upward to FFFFH has $\mathbf{A_{15}} = 1$. *Every* address in the bottom half, that is, 7FFFH down to 0000H has $\mathbf{A_{15}} = 0$. In general, the most significant address bit ($\mathbf{A_{n-1}}$ for an n-bit address bus) divides the memory exactly in half. In the 16-bit example, every address from C000H upward to FFFFH has address bit $\mathbf{A_{14}}$ (the second m.s.b.) equal to 1. Therefore all addresses in the top quarter of the memory map have $\mathbf{A_{15}}$ =1 and $\mathbf{A_{14}} = 1$. In general, the value of the two msb's $\mathbf{A_{n-1}}$ and $\mathbf{A_{n-2}}$ of any n-bit address thus identify which quarter of the memory map in which it belongs. Similarly, the memory map is divided into eighths by the first *three* msb's of the address and so on. Try a few values to convince yourself that this is true.

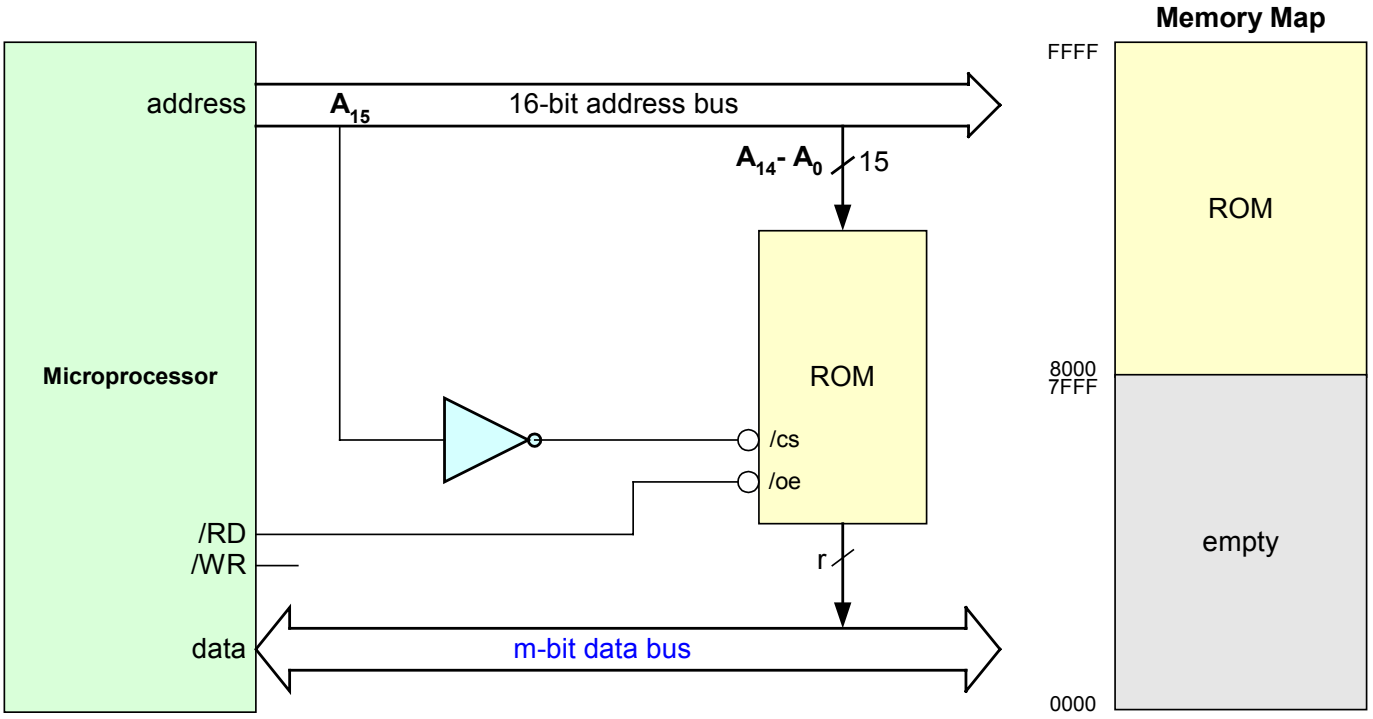
A Simple Example of Memory Decode Logic

A typical microprocessor memory system may contain several types of memory technologies including ROM (e.g. Flash), fast SRAM, high-density DRAM as well as empty space in which there is no memory available. The memory map shows the relative space occupied by all memory devices (and any empty space) proportionally to the total address space. The memory map may also indicate starting- and ending-addresses for each of theses regions as well as I/O device addresses if memory-mapped I/O is being used (however, we will not consider memory-mapped I/O in this discussion)

In most real implementations, the address space is rarely completely filled with memory devices. For example, in a typical PC, the Pentium processor has a 32-bit address bus which provides a 4G address space. Most desktop systems however, have 500Mbytes or less of memory which represents no more than 1/8 of the possible address space. That is, 7/8 (87.5 %) of the address space is empty ! Even with 1Gbyte of memory in your system, you are using only 25%.

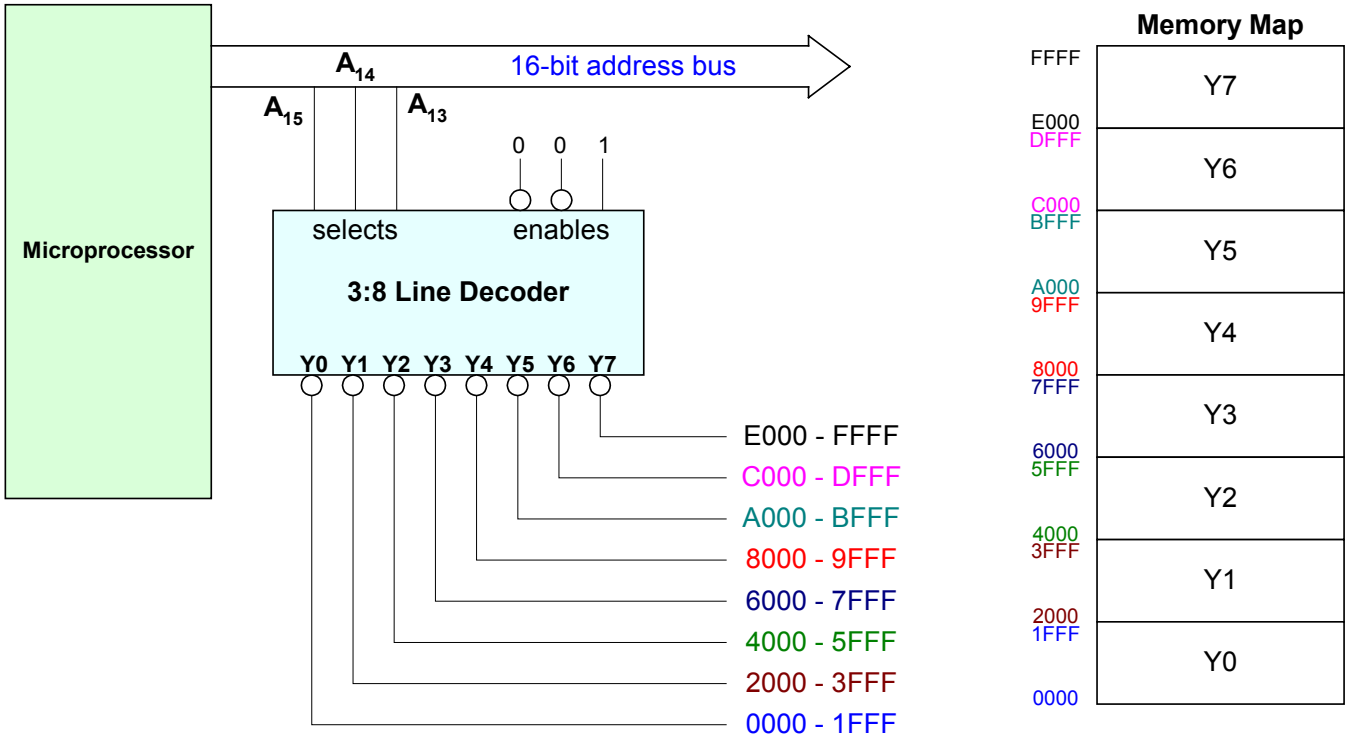
Think about this: *Next generation microprocessors will have 64-bit address buses. Estimate the cost at today's prices, to fully populate a 64-bit address space with DRAM.*

At this point, we can demonstrate a very simple decoding logic circuit – it is nothing more than an inverter whose input is connected to $\mathbf{A_{15}}$ as shown in the following diagram. When $\mathbf{A_{15}}$ = 1, the ROM device is enabled and responds to read cycles indicated by /RD = 0 for addresses 8000H to FFFFH. The ROM has 15 address lines and **r** data lines; therefore its organization is 32K x **r**. Note that decoding logic is independent of the value of **r**. If the microprocessor generates any address from 0000H to 7FFFH, the ROM is disabled and nothing will be read since the bottom half of the memory is empty ! The memory map shows this simple arrangement. A second 32K x **r** memory device could be added to populate the bottom half of the memory space by connecting its chip select directly to $\mathbf{A_{15}}$ (no inverter). This would then completely fill the 64K space.



A Second Example of a Memory Decode Logic

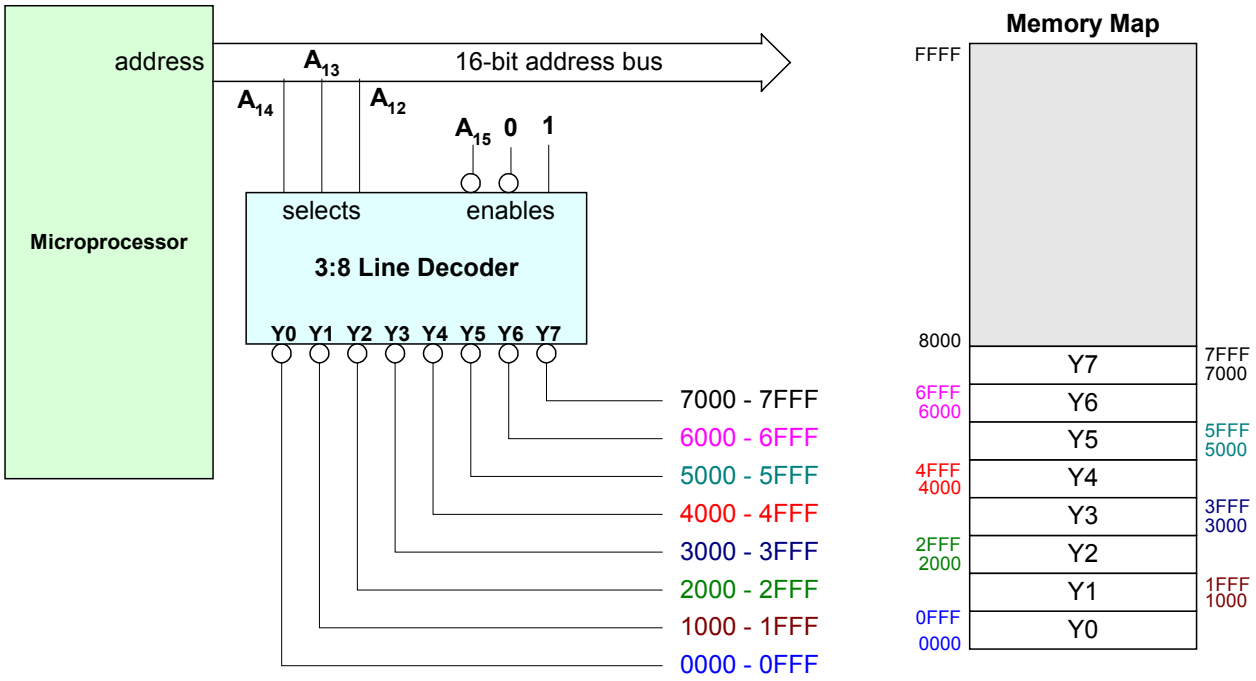
In the previous example, the inverter acts as a 1-to-2 decoder that responds to half of the address space. A binary decoder can be used to divide the address space by higher powers of 2. For example, in the figure below, the 3 msb's of an address bus $\mathbf{A_{15}}$, $\mathbf{A_{14}}$ and $\mathbf{A_{13}}$ are connected to the select inputs of a 3-to-8 decoder to divide the 64K address space evenly into eight 8K blocks. The decoder also has 3 enables which are fixed inputs as shown. The eight decoder outputs activate in the 8K address ranges as shown (in hexadecimal) which leads to the memory map as shown. LO-true outputs from the decoder are shown since most memory devices have LO-true chip select inputs.



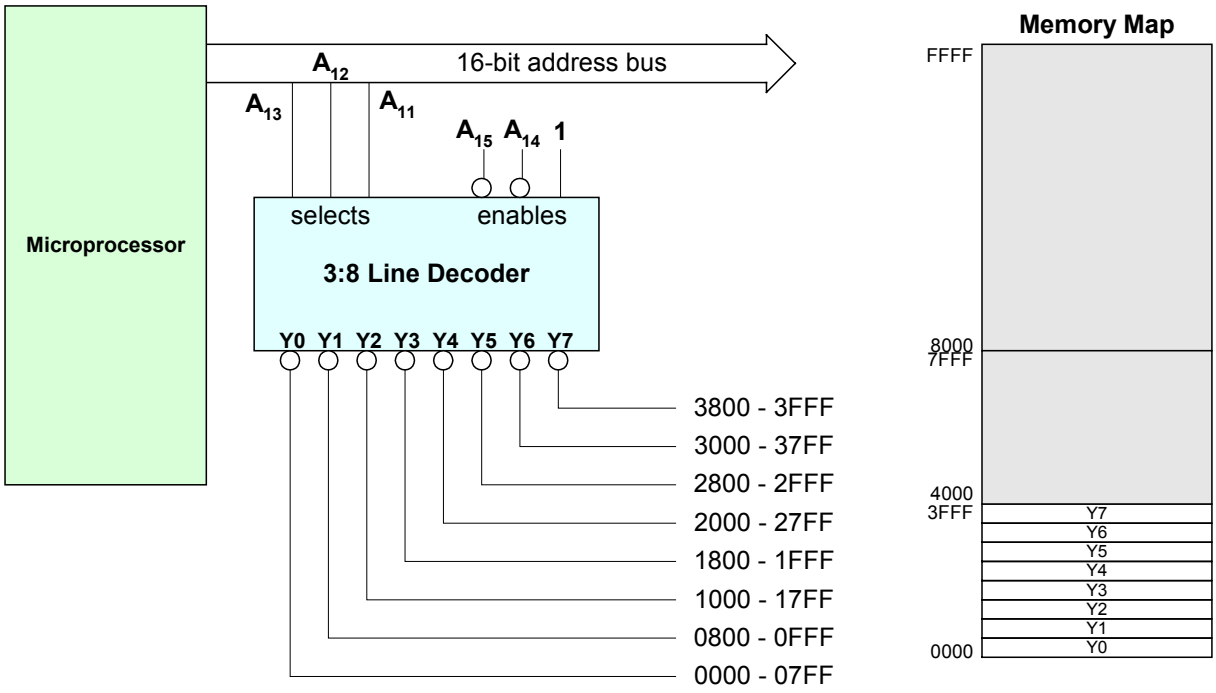
With this circuit, as many as 8 memory devices, each to a maximum size of 8K, could be connected to the address and data buses. The address range in which each device responds is, of course, determined by which of the decoder outputs is used to drive its chip select input. The regions of the memory map have been labelled with the corresponding names of the decoder outputs.

Using address bits to avoid parts of the Memory Map

Suppose now that A_{15} is used as an ***enable*** to the decoder and that the select inputs of the decoder are connected to bus A_{14} , A_{13} and A_{12} as shown in the next diagram. The affect on the memory map (also shown below) is to divide the bottom ***half*** of the address space into eight blocks each of 4K. Any address from the top half of memory (A_{15} =1) will produce no response from this decode logic since the decoder will be disabled. Only addresses with A_{15} = 0 will result in a response from the decoder. Up to 8 memory devices each of a maximum size of 4K could be connected to the address and data buses with this circuit.

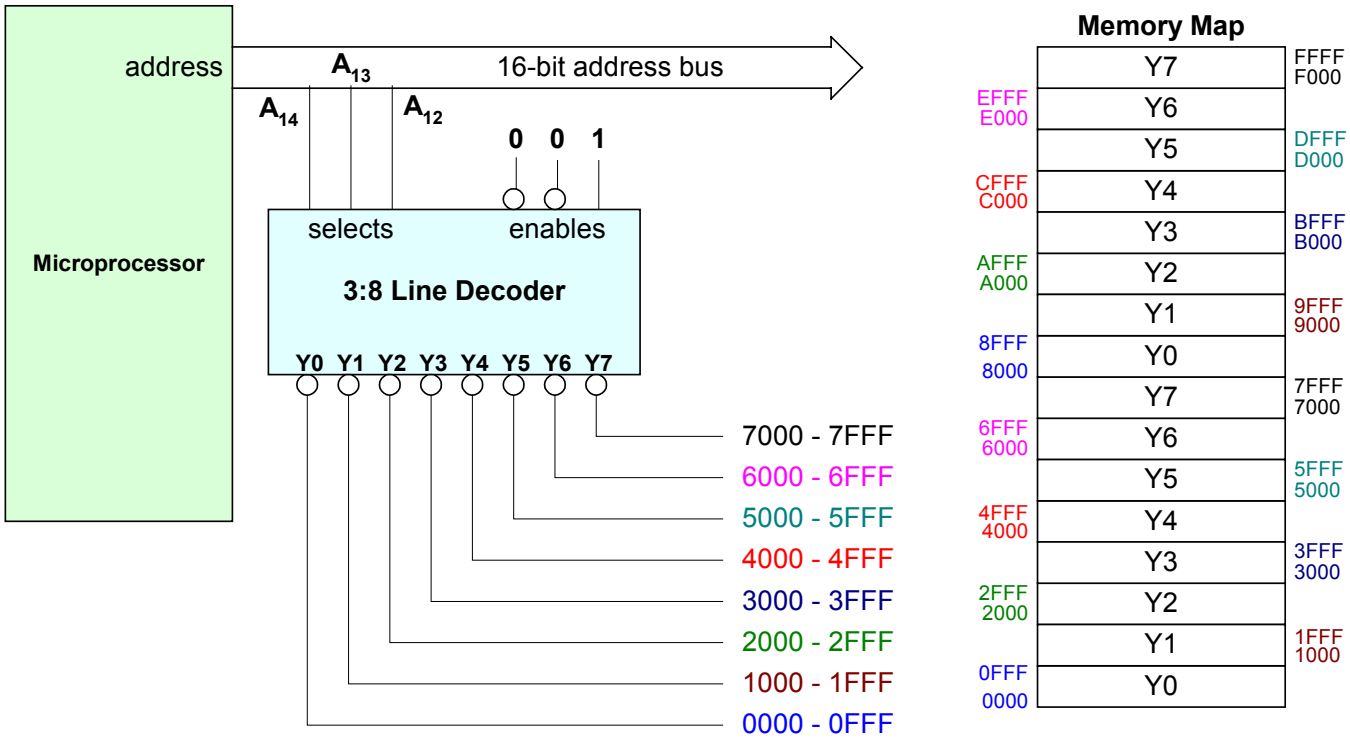


Similary, if both A_{15} and A_{14} are used as enables to the decoder, the bottom ***quarter*** of the address space is divided into eight 2K blocks with addresses as shown:



Non-exhaustive Memory Decode Logic

Consider the following example where, as in a previous example, the select inputs of the decoder are connected to bus $\mathbf{A_{14}}$, $\mathbf{A_{13}}$ and $\mathbf{A_{12}}$ as shown. However, now, the decoder is permanently enabled by fixing its enables to **0 0 1**. Address bit $\mathbf{A_{15}}$ is not connected anywhere in the decode logic and is effectively ignored. The effect on the memory map is to replicate each block of decoded space in the top and bottom half of the address space.

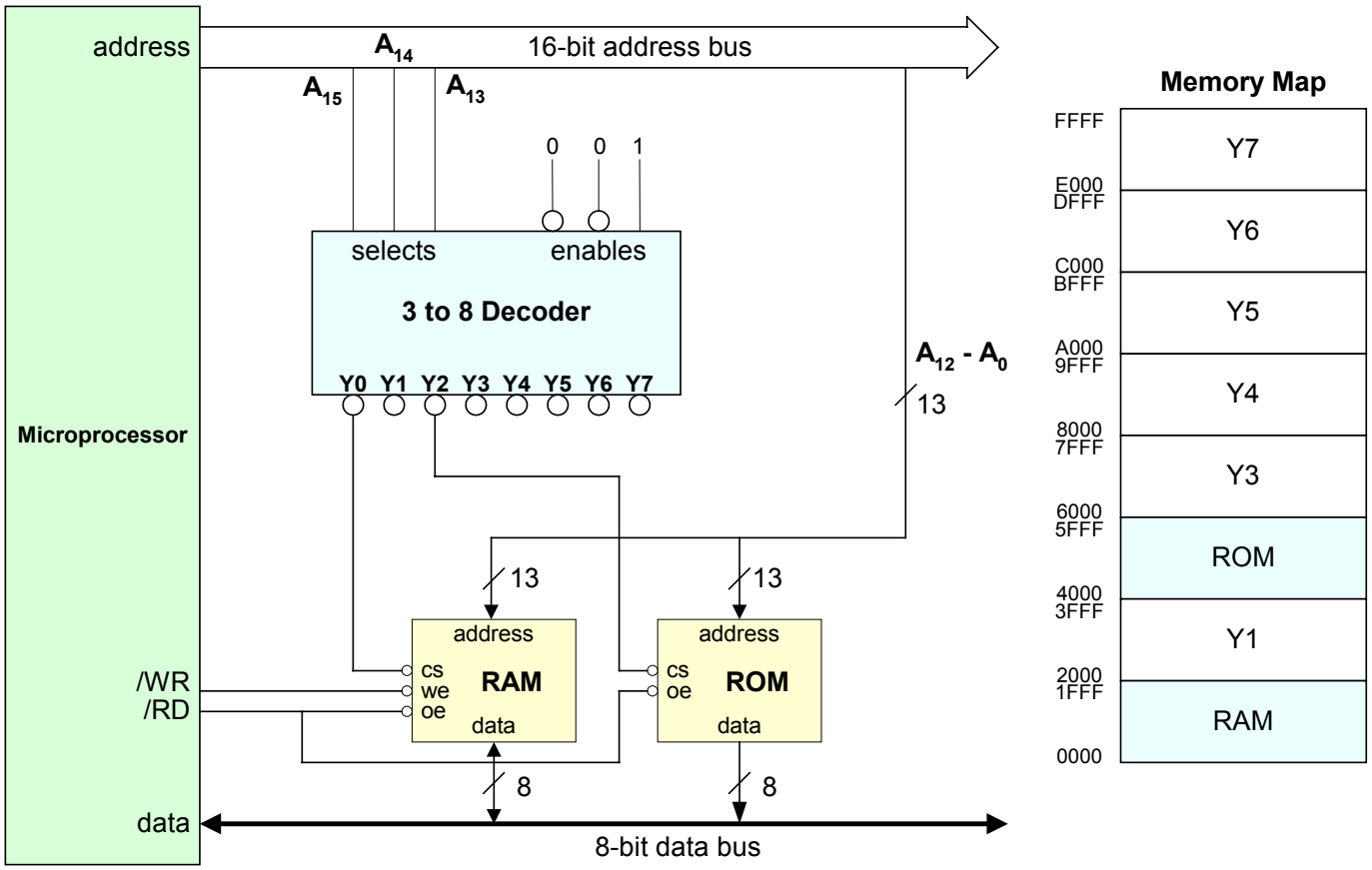


That is, there are ***two*** addresses, one with $\mathbf{A_{15}} = 0$ and one with $\mathbf{A_{15}} = 1$, that activate the same decoder output. For example, address 0000 and adresss 8000H each activate output Y0. Similarly, addresses 1000H and 9000H both activate output Y1 and so on. The result is that half of the memory addresses are wasted. In this example, there are only 32K unique addresses (0000 to 7FFFH). Addresses 8000H to FFFFH are not available for other uses. This is known as ***non-exhaustive*** decoding.

Of course $\mathbf{A_{15}}$ still exists on the address bus and is still being driven by the microprocessor. We haven't eliminated it, we are only ignoring $\mathbf{A_{15}}$ in the decode logic. Although the entire address space is used, there are only 8 unique address ranges for a total of 32K.

Adding Memory Devices

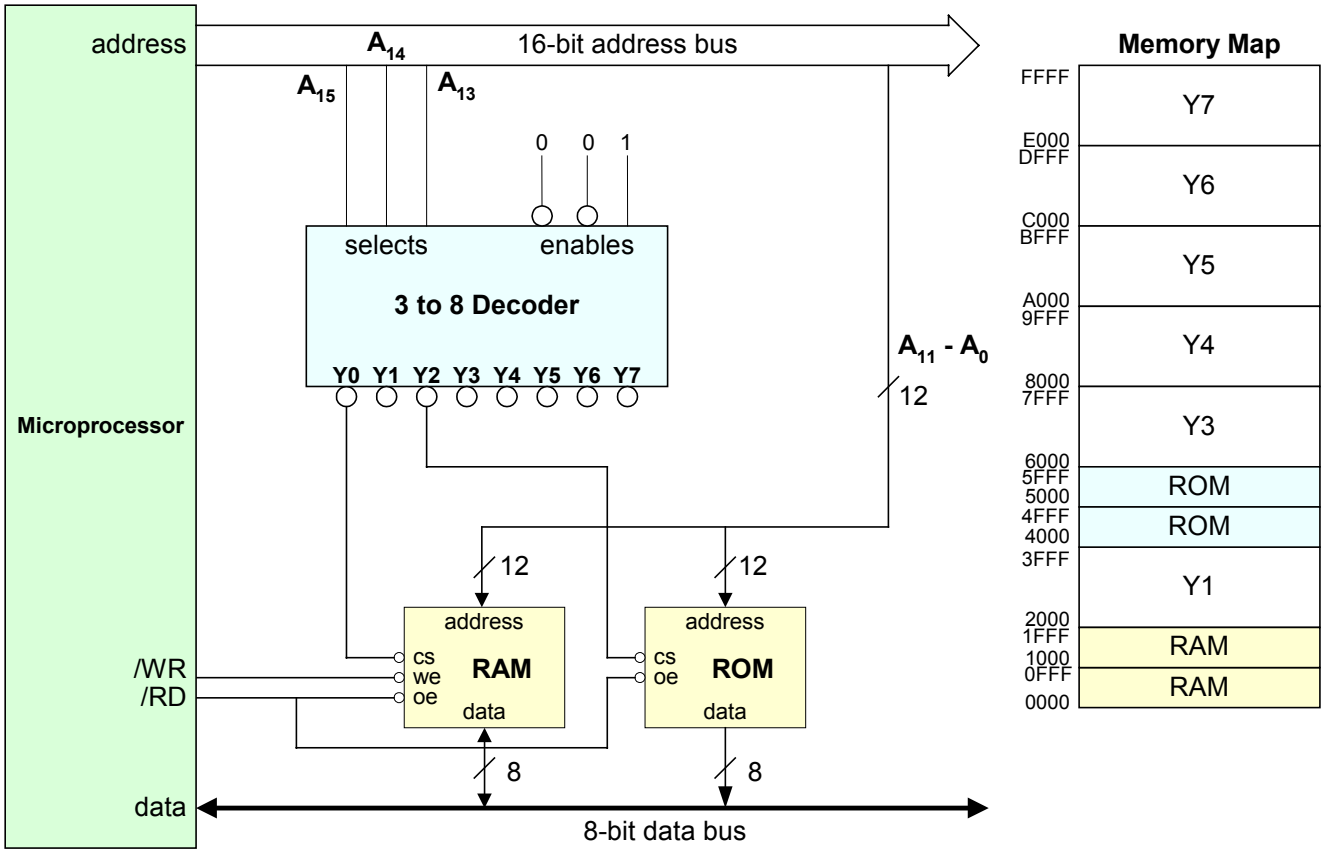
Returning to the example in which the address space is divided into eight 8K blocks, we extend the discussion by adding actual memory devices to the circuit as shown:



The RAM and ROM are each 8K x 8 devices (13 address lines and 8 data lines). The output enable (/OE) of each device is connected to the /RD signal so that the memory places its value onto the data bus at the proper time in the memory read machine cycle. The /WE input of the RAM is controlled by the /WR signal so that the memory takes a value from the data bus at the proper time in the memory write machine cycle. The address range to which each memory device responds is determined by the choice of connection from the decoder output to its chip select input. As shown, the RAM responds to addresses 0000 to 1FFFH and the ROM responds to addresses 4000H to 5FFFH. This is easily observed in the memory map. By changing its chip select connection to other unused decoder outputs, a memory device may be seen to reside in any of the 8K address blocks.

If the microprocessor reads or writes with any address in an 8K range in which no memory device is enabled, then no valid data is transferred. Up to eight 8K x 8 devices could be supported with this circuit. With the two memory devices as shown, the percentage of the address space that is populated is: (8K + 8K)/64K = 25%. There is decoding logic for the entire 64K space.

If the memory device is smaller than the decoded block, then regions of redundant redundant addresses are generated. For example, using the same decode logic as above, consider the following circuit in which 4K x 8 memory devices are used in place of the 8K memory devices:



Since address bit $\mathbf{A_{12}}$ is effectively not used, the memory map contains duplicate regions of addresses which access the same physical memory device ! For example, address 0000 accesses the same location as address 1000H, address 4FFFH accesses the same physical memory location as address 5FFFH and so on. Two 8K blocks are occupied in the address space, but there is only 4K + 4K = 8K of physical memory available. So the percentage of populated space is : (4K + 4K)/64K = 12.5%.

Address bit $\mathbf{A_{12}}$ is still driven by the microprocessor on the address bus but is ignored externally and again this leads to wasted addresses in the memory map as in the case on non-exhaustive decode logic. In general, logically replicated areas occur in the memory map when $\mathbf{k + p \neq n}$ or $\mathbf{k + q \neq n}$ (refer to the first diagram in this document). Redundant areas of duplicated address space as shown in this example, is sometimes called ***foldback memory***.

Foldback memory regions are transparent to the microprocessor since it operates independently of any external decode logic and memory devices. The microprocessor executes memory read and write cycles and relies on the design of the external logic to ensure that reads and writes from/to memory take place correctly.

Think about this: *If instead of 4K devices as above, 2K devices were used with this decode logic, what would be the effect in the memory map?*

Conclusion

This design of decode logic as shown here is valid for any size of address space, that is, any value of **n**. Decode logic is generally not dependent on the size of the data bus. There are many ways to implement decode logic; one of the most common uses of the early PAL devices was to implement decode logic for microprocessor memory systems.